Software Defined Networking for the Industrial Internet of Things

By

MICHAEL BADDELEY

Department of Electrical and Electronic Engineering
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

MARCH 2020

Word count: 45093

Abstract

While the increasing ubiquity of embedded devices has given rise to the 'smart' moniker applied to everyday objects, the underpinning wireless communication protocols collectively incorporate them into the Internet of Things (IoT). These protocols facilitate communication to, from, and between objects, and have spawned ever more sophisticated applications in both the home and in industry. While much of the publicity around this phenomenon has focused on domestic uses, a quiet revolution has been taking place within industrial and commercial sectors: as the initial promises of IoT are finally met by technological capability. It is here, in the Industrial Internet of Things (IIoT), where there is the opportunity to streamline operations, such as managing warehouse stock for Just-In-Time (JIT) manufacturing; support safety-critical systems through the complex monitoring of sensor and actuator networks; and offer new business models, where IoT infrastructure can be offered as a service (IoTaaS).

Traditionally, low-power wireless mesh networks have been at the forefront of the IoT conversation. However, the increasing complexity of these networks has laid bare the shortcomings inherent in current control architectures and protocols, where lossy channels and multi-hop topologies present significant challenges. To address these issues, there has been considerable interest in applying the concepts of Software Defined Networking (SDN), which over the past decade has liberated data centre and campus network management from reliance on vertical infrastructure practices. However, the centralised SDN approach faces considerable challenges in the constrained environments present in low-power wireless networks.

This thesis explores not only how SDN concepts can be used to provide dynamic and flexible control in IIoT, but crucially how to address and manage the SDN overhead. It presents analytical, simulated, and experimental results, as well as the design and implementation of two novel SDN architectures for low-power wireless networks: μSDN and Atomic-SDN. These results demonstrate both that the challenges of applying SDN within constrained IoT networks can be overcome, and that SDN can be used to address the complex and diverse traffic requirements of IIoT applications across low-power wireless networks. The synchronous flooding approach of Atomic-SDN, in particular, provides an effective means of achieving the one-to-many traffic pattern required in distributed control systems, and makes it a highly promising solution for deploying SDN within low-power wireless IIoT networks.

Dedication and acknowledgements

I would like to express thanks to my academic supervisors, Reza Nejabati and George Oikonomou, as well as my industrial supervisor Mahesh Sooriyabandara at Toshiba Research Europe Ltd. Each has helped guide and support me over these last few years, and their knowledge and experience has been invaluable.

I am also grateful for the support of my fellow doctoral students, researchers, and the staff at the University of Bristol. Not only within the Centre for Doctoral Training (CDT) in Communications, which supported this PhD financially, but also in the High Performance Networking (HPN) group. Particular thanks also goes to my colleagues Usman Raza, Aleksandar Stanoev, and Yichao Jin at Toshiba, whose collaborative efforts have helped support my research, and with whom I've spent many a long night desperately hacking at competition code. As well as providing invaluable feedback over the years, their friendship has helped maintain my sanity.

It would be remiss of me not to mention two of my closest friends and recent groomsmen at my wedding: Anthony Portelli and Tommy Thompson, whose own experiences and enthusiastic encouragement prompted me to embark on this PhD in the first place.

However, most importantly, none of this would have been possible without the eternal patience and constant support of my wife, Lauren. You've been at my side through every high and every low. You've picked me up through every crises of confidence, and pushed me to accomplish things I never thought I could achieve. I would never have made it to this point without you, and without you this journey would never have been the same.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

Table of Contents

List of Acronyms

5G Fifth Generation

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks

6TiSCH IPv6 over the TSCH mode of IEEE 802.15.4e

ACK Acknowledgement

AMI Advanced Metering Infrastructure

APB Abstract Protocol Builder

API Abstract Programming Interface

ASDS Adaptive Software Defined Scheduling

ASIC Application Specific Integrated Circuit

ASK Amplitude Shift Keying

BER Bit Error Rate

BGP Border Gateway Protocol

BLE Bluetooth Low Energy

BPSK Binary Phase Shift Keying

CAPEX Capital Expenditure

CBR Constant Bit-Rate

CCA Clear Channel Assessment

CFO Carrier Frequency Offset

CLI Command Line Interface

CMQ Control Message Quenching

CoAP Constrained Application Protocol

CoJP Constrained Join Protocol

CONF Configuration

COTS Commercial-Off-The-Shelf

CPS Cyber Physical Systems

CPU Central Processing Unit

CROWN Concurrent ReceptiOns in Wireless Sensor and Actuator Networks

CSMA Carrier-Sense Multiple Access

CSMA/CA Carrier-Sense Multiple Access / Collision Avoidance

CSS Chirp Spread Spectrum

CT Concurrent Transmission

CTS Clear to Send

DAG Directed Acyclic Graph

DAL Device Abstraction Layer

DAO Destination Advertisement Object

DetNet Deterministic Networking

DIO DODAG Information Object

DIS DODAG Information Solicitation

DODAG Direction-Orientated Directed Acyclic Graph

DSSS Direct Sequence Spread Spectrum

DTLS Datagram Transport Layer Security

EEPROM Electrically Erasable Programmable Read Only Memory

FAN Field Area Network

FCS Frame Check Sequence

FEC Forward Error Correction

ForCES Forwarding and Control Element Separation

FRR Flowtable Rule Refresh

FSM Finite State Machine

G-MPLS Generalised Multi-Protocol Label Switching

HAL Hardware Abstraction Layer

HC Header Compression

IaaS Infrastructure-as-a-Service

ICMPv6 Internet Control Message Protocol version 6

IETF Internet Engineering Task Force

IIoT Industrial Internet of Things

INPP In-Network Packet Processing

IoT Internet of Things

IoTaaS IoT-as-a-Service

IP Internet Protocol

IPv6 Internet Protocol version 6

IRTF Internet Research Task Force

JIT Just-In-Time

LLN Low-Power and Lossy Network

LoRa Long Range

LOS Line Of Site

LTE Long-Term Evolution

M2M Machine-to-Machine

MAC Medium Access Control

MCU Microcontroller Unit

MPDU MAC Protocol Data Unit

MPLS Multiprotocol Switch Labeling

MRHOF Minimum Rank Objective Function

MSK Minimum Shift Keying

MTC Machine-Type Communication

MTU Maximum Transmission Unit

NB Northbound

NB-IoT Narrowband IoT

ND Neighbour Discovery

NETCONF Network Configuration Protocol

NFC Near Field Communication

NFV Network Function Virtualisation

NFVRG Network Function Virtualisation Research Group

NOS Network Operating System

NSAL Network State Abstraction Layer

ODL OpenDaylight

OF Objective Function

OF0 Objective Function Zero

ONF Open Networking Foundation

ONOS Open Networking Operating System

OOB Out-Of-Band

OPEX Operational Expenditure

OQPSK Offset Quadrature Phase Shift Keying

OS Operating System

OSCORE Object Security for Constrained RESTful Environments

OVS Open vSwitch

OVSDB Open vSwitch Database

P2P Peer-to-Peer

PCE Path Computation Engine

PCEP Path Computation Element Communication

POF Protocol Oblivious Forwarding

PPDU PHY Protocol Data Unit

PPQ Partial Packet Queries

PRE Packet Replication and Elimination

PSSS Parallel Sequence Spread Spectrum

QoS Quality of Service

QPSK Quadrature Phase Shift Keying

RAM Random Access Memory

RAW Reliable and Available Wireless

RDC Radio Duty Cycling

RFID Radio-Frequency Identification

RH Routing Header

RLNC Random Layer Network Coding

ROM Read Only Memory

RPL Routing Protocol for Low-Power and Lossy Networks

RTS Request to Send

SB Southbound

SDN Software Defined Networking

SDWMN Software Defined Wireless Mesh Network

SDWN Software Defined Wireless Network

SDWSN Software Defined Wireless Sensor Network

SF Synchronous Flooding

SNMP Simple Network Management Protocol

SNR Signal to Noise Ratio

SOC System-on-Chip

SOTA State-of-the-Art

SR Source Routing

SRH Source Routing Header

SRI Source Routing Injection

SUN Smart Utility Network

TCAM Ternary Content Addressable Memory

TCP Transmission Control Protocol

TDMA Time Division Multiple Access

TSCH Time Scheduled Channel Hopping

UDGM Unit Disk Graph Medium

UDP User Datagram Protocol

UWB Ultra Wide Band

VBR Variable Bit-Rate

VNF Virtual Network Function

VXLAN Virtual Extensible LAN

WG Working Group

WLAN Wireless Local Area Network

WMN Wireless Mesh Network

WNAN Wireless Neighbourhood Area Network

WNAN Wireless Wide Area Network

WPAN Wireless Personal Area Network

WSDN Wireless Software Defined Network

WSN Wireless Sensor Network

XMPP Extensible Messaging and Presence Protocol

NOTATION

μ Micro

GB Gigabytes

GHz Gigahertz

KB Kilabytes

MB Megabytes

MHz Megahertz

mW Milliwatts

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

The advent of programming languages, kernels, and operating systems have abstracted the complexities of managing low-level computer resources; so that, today, applications can be written, compiled, and successfully run without knowledge of the underlying processes or hardware. These tools introduced the means to manage, configure, and repurpose technology as needs and requirements change. This concept, when applied to networks, is known as Software Defined Networking (SDN).

Similar to the operating system on a computer, the architectural concept behind SDN gives us the power to abstract the complex management of network resources, provision these resources for multiple concurrent applications or operators, and easily roll-out new functionality. These tools are increasingly seen as critical to the management of the embedded networks which collectively constitute the Internet of Things (IoT).

However, as IoT solutions are leveraged to solve progressively difficult problems, deploying and maintaining such networks is becoming increasingly complex. With industry increasingly seen as the driver of IoT growth [10], applications often need to meet stringent reliability, latency, and energy-efficiency requirements, and large-scale networks can consist of tens, to many thousands of nodes. Not only does this pose a significant resource management and allocation problem, it also represents considerable infrastructure investment. Yet although SDN is increasingly being used to solve similar issues in wired networks, the application of SDN within the constrained environment of low-power wireless faces significant hurdles: where the realities of limited radio, computing, and network resources clash with the low-latency and high throughput requirements typical of traditional SDN approaches.

This introductory chapter outlines the important role that low-power wireless networks have traditionally played within industry, first as Wireless Sensor Networks (WSNs), and later

in IoT. It then highlights the recent drive towards network 'softwarization', introducing the concept of SDN and the considerable role it has played in recent network research. Laterally, it advocates exploring SDN concepts as a means of addressing the complexities and challenges posed by future Industrial IoT networks, before examining how the the constraints of low-power wireless shape this approach. Finally, this chapter presents this author's contributions to current State-of-the-Art (SOTA) and the overall thesis structure.

1.1 The Role of Low-Power Wireless Networks in Industrial IoT

The number of devices participating in the Internet of Things (IoT) is expected to grow to 14.6 billion connections by 2022, with much of this growth driven by demand from industrial sectors such as energy and manufacturing [10]. Low-power wireless sensor and actuator networks, commonly employed in these industries, have traditionally dominated the conversation of how IoT networks are implemented and maintained. Operating on the unlicensed 2.4 GHz or sub-GHz bands, their low-cost and low barrier-to-entry has made them ideal for data collection and monitoring applications: where they are used to collect sensor readings over long periods, and funnel the data towards a central gateway for later processing on backend systems. As such, they have played a crucial role in the emergence of Industry 4.0 [11] and Industrial IoT (IIoT), where the innovative deployment of embedded low-cost wireless networks can provide key data to help manage intricate systems and streamline business practices.

Although a number of new IoT standards have been proposed and implemented as part of the push towards 5G, many of these solutions involve the installation of cost-prohibitive infrastructure, or expend considerably more energy than current and emerging low-power wireless (such as IEEE 802.15.4 [1], as well as the more recent Bluetooth Low Energy (BLE) [12] and LoRaWAN [13] standards). These properties are a fundamental requirement in large-scale IIoT networks, such as Smart Utility Networks (SUNs) and Advanced Metering Infrastructure (AMI), which can involve the deployment of hundreds of thousands, or even millions of nodes [14], and can contain battery powered devices that need to operate over long periods. Low-power wireless, in particular IEEE 802.15.4, will therefore continue to play an important role in the future of IoT and is the focus of this thesis, as outlined in Figure 1.1.

While the traditional approach to IoT over the past decade has been driven primarily by data collection services, and inferring value from that data, recent efforts in low-power wireless research aim to support more dynamic and complex scenarios in two ways. Firstly, at the level of the local mesh, networks need to be able to provide deterministic Quality of Service (QoS) guarantees to traffic, moving away from 'best-effort' approaches. Work to address this is being undertaken by standardisation activities such as 6TiSCH [2], which employs spatial and frequency diversity in the wireless mesh to provide latency guarantees and 'five-nines' (99.999%) reliability for large-scale IEEE 802.15.4-2015 [1] industrial networks. Additionally,

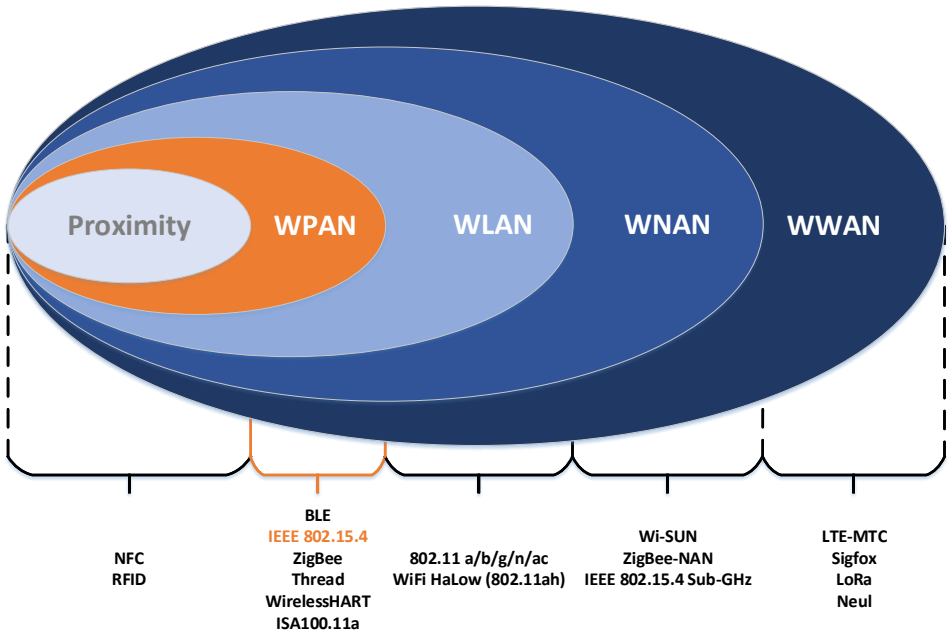


Figure 1.1: Summary of IoT communication technologies with respect to application area: Proximity (Body) to Wireless Wide Area Networks (WWAN). As highlighted in orange, this thesis focuses on IEEE 802.15.4 Wireless Personal Area Networks (WPAN) commonly used in industrial mesh networking.

recent research has shown that protocols based on Concurrent Transmissions (CTs) [6] can synchronously flood the mesh network to achieve similar reliability results at theoretical lower bounds of latency.

Secondly, there is a growing movement away from single-application models, and a case can be made that the future of IIoT is in networks that can provide service guarantees across multiple applications and multiple tenants. In SUN and AMI applications, for example, there is interest in monetising the under-utilisation of the extensive IoT infrastructure. As throughput can often be measured in terms of packets per day, week, or even months, there is considerable argument for opening up the network to other applications or processes, as long as the requirements and service level agreements can be guaranteed.

Underpinning both requirements is the need for a new control solution, which is able to make informed decisions at the macro level based on the needs of higher-level applications, and can apply these locally based on the the current availability and slicing of limited network and spectrum resources.

1.2 The Drive Towards 'Network Softwarization' and 'Programmable Networks'

In recent years a popular topic in networks research has been exploring how 'softwarization' can transform the traditional vertical model of network infrastructure into something that is more dynamic, and less reliant on proprietary, application-specific, and costly hardware. Aside from its somewhat cumbersome idiom, at its core network 'softwarization' represents a very simple concept: that all aspects and functions of the network should be as programmable and easy to manage or update as software on a computer. Just as a computer programmer shouldn't necessarily need to painstakingly manage processor instructions and memory registers across different hardware platforms, nor should one need to worry about the specifics of the underlying infrastructure when implementing or deploying network functions.

The most well known term associated with this concept is Software Defined Networking (SDN), which, at its most simple, encompasses the idea that all network devices can be reduced to dumb switches (rather than relying on proprietary and dedicated hardware) that can in turn be configured to perform any manner of network functions by an all-knowing central controller. Anything from routing, to load-balancing, to firewall rules are deployed at the controller, and then 'programmed' into the network through an abstraction layer, transmitting simple configuration instructions that control the flow of data across individual devices. This powerful concept, shown in Figure 1.2, allows network decisions to be taken with consideration to the global network state, rather than taking them locally on the device. It moves the view of network architecture from one where configuration is performed across multiple vertical silos, to managing the entire network from a dynamic, horizontal control plane that can adapt to changing traffic requirements.

As it has moved from research and into industry, SDN has been immensely successful in upending the data centre and cloud computing markets [15–18]. This success has been the basis for a revolution in how we design, implement, and think about networks. Driving the push towards network 'softwarization' and 'programmable networks', research in this area has spurred renewed interest in concepts such as *Network Function Virtualisation (NFV)*, *Network Slicing*, and *Heterogeneous Networks*. By providing an abstracted network view and a framework for virtualising network functions, SDN allows services to be centrally programmed onto functionally agnostic hardware. The ability to reconfigure the network as needed, quickly install new protocols, or slice network resources across applications and tenants, allows networks to adapt to changing requirements or shifts in business needs. Implementation of this concept has been leveraged for key-use cases by a number of leading technology firms [19]. Yet this success is fundamentally supported by the availability of high data rate and dedicated out-of-band connections in traditional wired and optical networks [4]; thus enabling reliable, low-latency links between a centralised controller (or a number of distributed controllers) and SDN switches. This makes configuration tasks, such as installing flowtable rules on multiple devices, both extremely responsive and highly

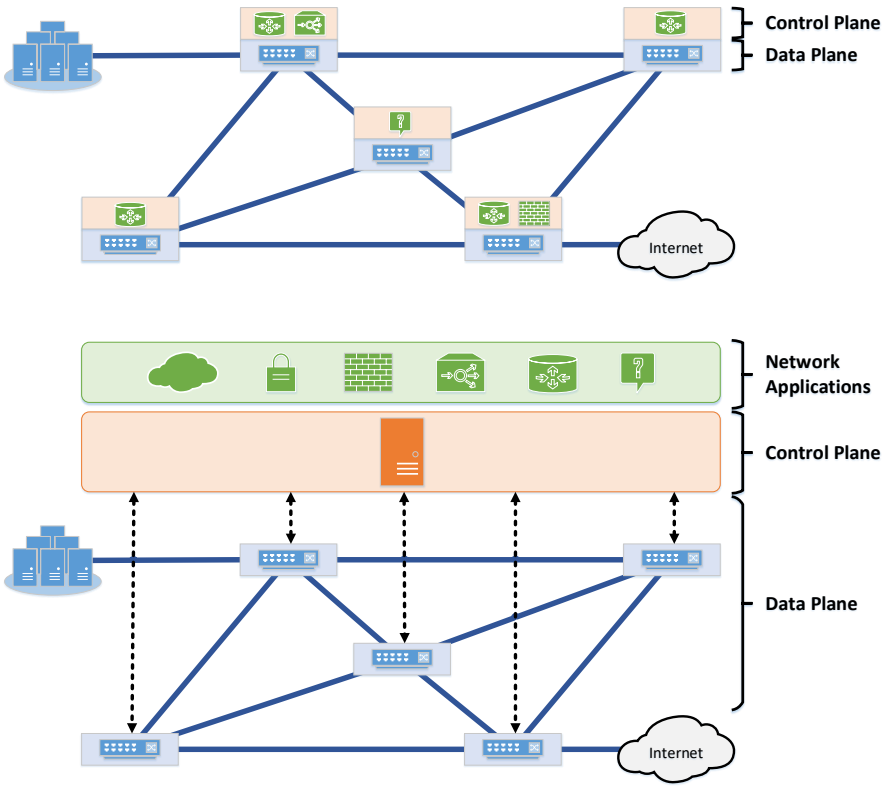


Figure 1.2: Traditional vertical (upper) vs. horizontal SDN (lower) approach to network architecture.

scalable.

1.3 Extending SDN to Future Industrial IoT Networks

As embedded devices increasingly blur the lines between the digital and physical worlds, there has been a keen interest in extending the SDN concept to manage the rapid growth and increasing complexity of low-power IoT networks [4, 19–23]. Although the control separation concept that lies at the heart of the SDN is a simple one, it is an enabler for the network and application virtualisation concepts that have helped power the cloud computing revolution of the previous decade. If SDN can be harnessed within constrained mesh networks that lie at the very edge of IoT, it can not only offer a sophisticated toolbox to manage applications across highly complex environments, but also an opportunity to move away from single-application infrastructure and abstract this complexity into a single heterogeneous network: from backhaul servers to embedded sensors.

However, the shared nature of the underlying wireless medium, multi-hop links, duty-cycling and power restrictions, and stringent constraints on network resources pose significant additional challenges not present in SDN's traditional application within wired and optical networks.

Furthermore, as a centralised control architecture, SDN requires frequent back-and-forth communication between the controller(s) and network nodes. The traffic follows a variety of different communication patterns including *many-to-one*, *one-to-many* and *one-to-one*, however standard low-power protocols such as the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL) [24], provide less-than-optimal performance for the plurality of these traffic patterns, and the challenge of managing this additional SDN overhead across the low-power mesh is considerable.

This issue of overhead becomes increasingly acute as the mesh scales, and attempts to directly map aspects of traditional SDN architecture to low-power wireless becomes an extremely complex operation. This complexity fundamentally arises from the controller not only having to communicate reliably with all nodes, but that each individual operation (for example, to set a path between two nodes) can mean the replication of control messages across multiple nodes in order to correctly configure the network [25–27, C1]. The cost of servicing this overhead is an anathema to low-power wireless mesh networks, where both applications and control protocols must contend over extremely limited resources. These challenges are considerable, and if the benefits of SDN are to be delivered in low-power wireless networks a new approach is needed.

1.4 Contributions to State of the Art

This PhD, sponsored by Toshiba Research Europe Ltd, was started in 2015 with the aim of exploring the application of SDN architectural concepts within emerging IIoT scenarios such as AMI, smart cities, and factory automation and control. This thesis focuses on the IEEE 802.15.4 low-power wireless mesh standard (specifically, IEEE 802.15.4-2015 [1]) commonly employed within these networks. It provides detailed and comprehensive analysis of the challenges that SDN faces in the constrained environments present within low-power mesh networks, and proposes, implements, and evaluates novel solutions to overcome these challenges. This research has been driven by real-world problems and challenges experienced by the industrial sponsor when deploying extremely large SUN and AMI networks [14]. Table 1.1 provides a summary of research output from this PhD, and links each item to relevant research questions and chapters.

1.4.1 Research Questions and Objectives

This thesis addresses the following fundamental research questions on how SDN concepts can be applied within the scope of low-power wireless networks for Industrial IoT:

[Q1]: *What are the fundamental features of a minimal SDN solution?*

[Q2]: *How does SDN control signalling affect the low-power wireless mesh?*

[Q3]: *Are there solutions to reduce SDN overhead?*

[Q4]: *Can SDN scale in a low-power wireless mesh?*

[Q5]: *What advantages does SDN bring to a low-power wireless mesh?*

In line with the research questions above, this thesis aims to achieve the following objectives:

1. To outline how programmable network architectures can benefit future IIoT networks.
2. To clearly define the key challenges faced by SDN in low-power wireless mesh networks.
3. To propose methods and techniques to address these challenges.
4. To design and implement novel SDN architectures for low-power wireless mesh networks.
5. To evaluate these architectures against current SOTA approaches.
6. To demonstrate how SDN concepts can provide programmable mesh control.
7. To identify key research paths and future directions for SDN in IIoT.

1.4.2 Summary of Contributions

The following contributions have been identified from the research questions objectives defined in the previous section, and as a result of the research carried out during the course of this PhD. Contributions are listed by chapter, and are linked to the published works.

Contributions in Chapter 3 (from [C1, C4, R1]):

Results from this chapter were presented at IEEE Netsoft 2018.

- Established a lexicon for SDN functionality in low-power wireless and identified a minimal set of functionality needed to support SDN principles in multi-hop mesh networks.
- Introduced a novel SDN control link and controller discovery mechanism through coexistence with the Routing Protocol for Low-Power and Lossy Networks (RPL).
- Introduced a distributed technique to reduce SDN overhead by monitoring flow activity and increasing the timers of active flows.
- Designed and implemented μSDN, an optimised SDN stack for low-power IPv6 over IEEE 802.15.4 networks. This architecture presents a comprehensive and highly flexible SDN layer for low-power wireless networks. μSDN slots easily into the 'IoT stack' typically employed in IEEE 802.15.4 networks, and integrates seamlessly with existing control protocols such as RPL and IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN).
- Designed and implemented Atom, an embedded controller for μSDN that turns the RPL root node into an SDN controller capable of centralised SDN control decisions.

- Performed simulations showing that μSDN exhibits minimal additional overhead compared to other SDN implementations for low-power wireless, while providing a platform for virtual network functions and dynamic control of the mesh. The code repository for this architecture has been made publicly available.
- Demonstrated SDN as a method for network slicing in low-power mesh networks, showing that SDN can be used to serve traffic from multiple applications on a per-flow basis, and successfully re-routing critical traffic around heavily interfered nodes in a multi-hop mesh network.

Contributions in Chapter 4 (from [C2]):

Results from this chapter were presented at IEEE NFV-SDN 2017.

- Extended μSDN to integrate with the 6TiSCH stack.
- Performed simulations characterising SDN control traffic in a 6TiSCH network.
- Proposed and and implemented an algorithm for establishing 6TiSCH tracks for SDN control paths, establishing deterministic and isolated links between SDN nodes and a central control entity in IEEE 802.15.4-2015 Time Scheduled Channel Hopping (TSCH) networks.
- Performed simulations to evaluate this technique and compare against SDN network employing a standard 6TiSCH scheduling function.

Contributions in Chapter 5 (from [J1, C3, P1, DC1, DC2]:

Results from this chapter have been published in IEEE Access, and were presented at IEEE EWSN 2019.

- Illustrated key aspects of Synchronous Flooding (SF) and highlighted these as solutions to overcome challenges faced by SDN in low-power wireless.
- Proposed and designed an innovative method for dynamically constructing multiple different Synchronous Flooding (SF) protocols as traffic requirements change. This has been submitted as US patent no. 16/176659 on the 31st October 2018, which can be found in Appendix B.
- Designed and implemented Atomic-SDN, a middleware framework utilising SF as a reliable and scalable mechanism to concurrently broadcast to the entire local mesh. Atomic-SDN uses the concept proposed in [P1] to dynamically configure and schedule multiple synchronous flooding protocols, and matches these protocols to SDN services.
- Designed and implemented SF based protocols for SDN *collection*, *configuration*, and *reaction* services.

- Introduced a technique to perform slot-by-slot random channel hopping within a time synchronised multi-hop mesh network.
- Showed, through analysis and simulation, that an SF based SDN control plane delivers theoretical minimal bounds of latency for SDN control traffic *from* the controller, and improves latency for traffic *to* the controller by orders-of-magnitude compared to the current SOTA.
- Performed simulations demonstrating the scalability of Atomic-SDN within the local mesh, in comparison to current SOTA approaches.
- Performed testbed evaluation of Atomic-SDN, demonstrating robustness under interference through the injection of packet drops.
- Designed and implemented *Adaptive Software Defined Scheduling* (ASDS), a multi-protocol SF solution for dynamic wireless control application based on Atomic-SDN. This implementation was based on the Atomic-SDN architecture, and demonstrates a single SF stack capable of facilitating multiple traffic patterns across a low-power wireless mesh.
- Examined the reproducibility existing SOTA technique [28] to perform clock offset estimation for the unstable MSP430 oscillator, in order to mitigate clock frequency deviations across a multi-hop network. This technique was then extended so that estimation can be made on-the-fly for multiple different packet sizes.
- Introduced a method for surviving extremely high WiFi interference in a multi-hop mesh network through aggressive temporal and frequency diversity.
- Proposed and implemented SF based *collection* and *dissemination* protocols that exhibit minimal latency.
- Performed extensive evaluation of these protocols on a multi-hop mesh testbed.
- Participated in the IEEE EWSN Dependability Competition, winning 2^{nd} place in both *data collection* and *data dissemination* categories in 2019. This entry was the only solution to place in both categories, and the only one to achieve 100% reliability under extremely maximum interference levels. Full results and competition details can be found online [29], while selected results can be found in Appendix A.

Table 1.1: Summary of publications, competition entries, patents, and open-source code contributions; the research questions addressed, and the chapter in which they are discussed.

PhD Output		Research Questions	Relevant Chapter	Status
M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, "Evolving SDN for Low-Power IoT Networks," in 2018 IEEE Conference on Network Softwarization (NetSoft), June 2018.	[C1]	Q1,Q2,Q3	3	Published
M. Baddeley, R. Nejabati, G. Oikonomou, S. Gormus, M. Sooriyabandara and D. Simeonidou, "Isolating SDN control traffic with layer-2 slicing in 6TiSCH industrial IoT networks," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Nov. 2017.	[C2]	Q2,Q3,Q4	4	Published
M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, "Poster: Atomic-SDN: A Synchronous Flooding Framework for SDN Control of Low-Power Wireless," in 2019 International Conference on Embedded Wireless Systems and Networks (EWSN), Feb. 2019.	[C3]	Q3	5	Published
D. Saha, M. Shojaee, M. Baddeley, I. Haque, "An Energy-Aware SDN/NFV Framework for the Internet of Things," IFIP Networking, Jun. 2020.	[C4]	Q5	3/6	Accepted
M. Baddeley, U. Raza, G. Oikonomou, R. Nejabati, M. Sooriyabandara and D. Simeonidou, "Atomic-SDN: Is Synchronous Flooding the Solution to Software Defined Networking in IoT?," IEEE Access, May. 2019.	[J1]	Q3,Q4,Q5	5	Published
U. Raza, Y. Jin, A. Stanoev, M. Baddeley and M. Sooryiabandara, "Competition: CROWN Concurrent ReceptiOns in Wireless Sensor and Actuator Networks," in 2018 International Conference on Embedded Wireless Systems and Networks (EWSN), Feb. 2018.	[DC1]	Q5	5	Published
M. Baddeley, A. Stanoev, U.Raza, Y.Jin, and M. Sooriyabandara, "Competition: Adaptive Software Defined Scheduling of Low Power Wireless Networks," in 2019 International Conference on Embedded Wireless Systems and Networks (EWSN), Feb. 2019.	[DC2]	Q5	5	Published
M. Baddeley, U. Raza "A Controller for, and a Method of Processing Data Over, a Low-Power Wireless Software Defined Networking SDN Architecture" US Patent Application, no 16/176659, 2018	[P1]	Q3	5	Filed
M. Baddeley "μSDN: A low-overhead SDN stack and embedded SDN controller for Contiki" https://github.com/mbaddeley/usdn	[R1]	Q1,Q2,Q3	3	Online

1.4.3 Published Work

The following work has been published as an outcome of this PhD.

[C1]: M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, “Evolving SDN for Low-Power IoT Networks,” in 2018 IEEE Conference on Network Softwarization (NetSoft), June 2018.

[C2]: M. Baddeley, R. Nejabati, G. Oikonomou, S. Gormus, M. Sooriyabandara and D. Simeonidou, "Isolating SDN control traffic with layer-2 slicing in 6TiSCH industrial IoT networks," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Nov. 2017.

[C3]: M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, "Poster: Atomic-SDN: A Synchronous Flooding Framework for SDN Control of Low-Power Wireless," in 2019 International Conference on Embedded Wireless Systems and Networks (EWSN), Feb. 2019.

[J1]: M. Baddeley, U. Raza, G. Oikonomou, R. Nejabati, M. Sooriyabandara and D. Simeonidou, "Atomic-SDN: Is Synchronous Flooding the Solution to Software-Defined Networking in IoT?," *IEEE Access*, May. 2019.

In addition to the publications above, the following abstracts have been published in conjunction with the Dependability Competition at the International Conference on Embedded Wireless Systems and Networks (EWSN). Based on work in [J1], Adaptive Software Defined Scheduling [DC2] competed against 13 other teams consisting of 72 researchers, and won 2^{nd} place in both categories (*Data Collection* and *Data Dissemination*) of the 2019 competition.

[DC1]: U. Raza, Y. Jin, A. Stanoev, M. Baddeley and M. Sooryiabandara, "Competition: CROWN Concurrent ReceptiOns in Wireless Sensor and Actuator Networks," in 2018 International Conference on Embedded Wireless Systems and Networks (EWSN), Feb. 2018.

[DC2]: M. Baddeley, A. Stanoev, U.Raza, Y.Jin, and M. Sooriyabandara, "Competition: Adaptive Software Defined Scheduling of Low Power Wireless Networks," in 2019 International Conference on Embedded Wireless Systems and Networks (EWSN), Feb. 2019.

1.4.4 Submitted for Peer Review

The following work has been submitted for peer review.

[C4]: D. Saha, M. Shojaee, M. Baddeley, I. Haque, "An Energy-Aware SDN/NFV Framework for the Internet of Things" 2020 IEEE International Conference on Computer Communications (INFOCOM), Apr. 2020. [1]

1.4.5 Patents

The following patents have been filed and are pending.

[P1]: M. Baddeley, U. Raza "A Controller for, and a Method of Processing Data Over, a Low-Power Wireless Software Defined Networking SDN Architecture" *US Patent Application*, no 16/176659, 2018

1.4.6 Publicly Available Code Repositories

The following code has been made publicly available. It has used for teaching purposes at Dalhousie University (Halifax, Canada) and was used in submitted publication [C4].

[R1]: M. Baddeley, "μSDN: A low-overhead SDN stack and embedded SDN controller for Contiki", *https://github.com/mbaddeley/usdn*

1.5 Structure of Thesis

The remainder of this thesis is structured as follows, with the technical chapters summarised visually in Figure 1.3 to provide the reader with a view of this thesis in context of wider IoT research activities.

Chapter 2 provides the reader with background material on SDN and low-power wireless protocols, introduces necessary material on CT and SF, and provides a comprehensive overview of recent research exploring SDN in low-power wireless networks.

Chapter 3 presents the design and implementation of μSDN, a low-overhead SDN architecture for IEEE 802.15.4. As well as evaluating the performance of this solution against current IEEE 802.15.4 control protocols, this chapter investigates how μSDN can be used to slice limited network resources and provide dedicated control channels for SDN traffic.

Chapter 4 explores the additional challenges in migrating SDN based control architecture to IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) industrial networks, and presents the design and implementation of an extension to the μSDN architecture. This extension demonstrates

[1] This author has only had limited contribution towards this publication and, as such, is listed merely as reference to support the use of μSDN [R1] for further academic research.

how separating SDN control using dedicated network slices can isolate SDN control overhead from affecting other network traffic.

Chapter 5 explores how utilising Synchronous Flooding (SF) at the MAC layer can overcome many of the challenges outlined in previous chapters, and covers the design and implementation of Atomic-SDN, a temporally decoupled SDN solution based on Concurrent Transmissions (CT). This novel cross-layer architecture provides a framework capable of dynamic configuration and scheduling of CT-based SF protocols depending on current SDN control requirements, and results are presented showing how this approach is able to overcome the challenges faced by other SDN architectures. This chapter also details the design and implementation of a high-reliability SF solution based on Atomic-SDN; proposing and implementing multiple mechanisms in order to survive high levels of WiFi interference, and presenting metrics gathered as part of the IEEE EWSN Dependability Competition in order to benchmark the Atomic-SDN solution.

Final conclusions are presented in Chapter 6, detailing how the work presented in this PhD thesis overcomes many of the challenges in applying SDN concepts to low-power wireless networks, with additional recommendations of how it might be used within further research activities.

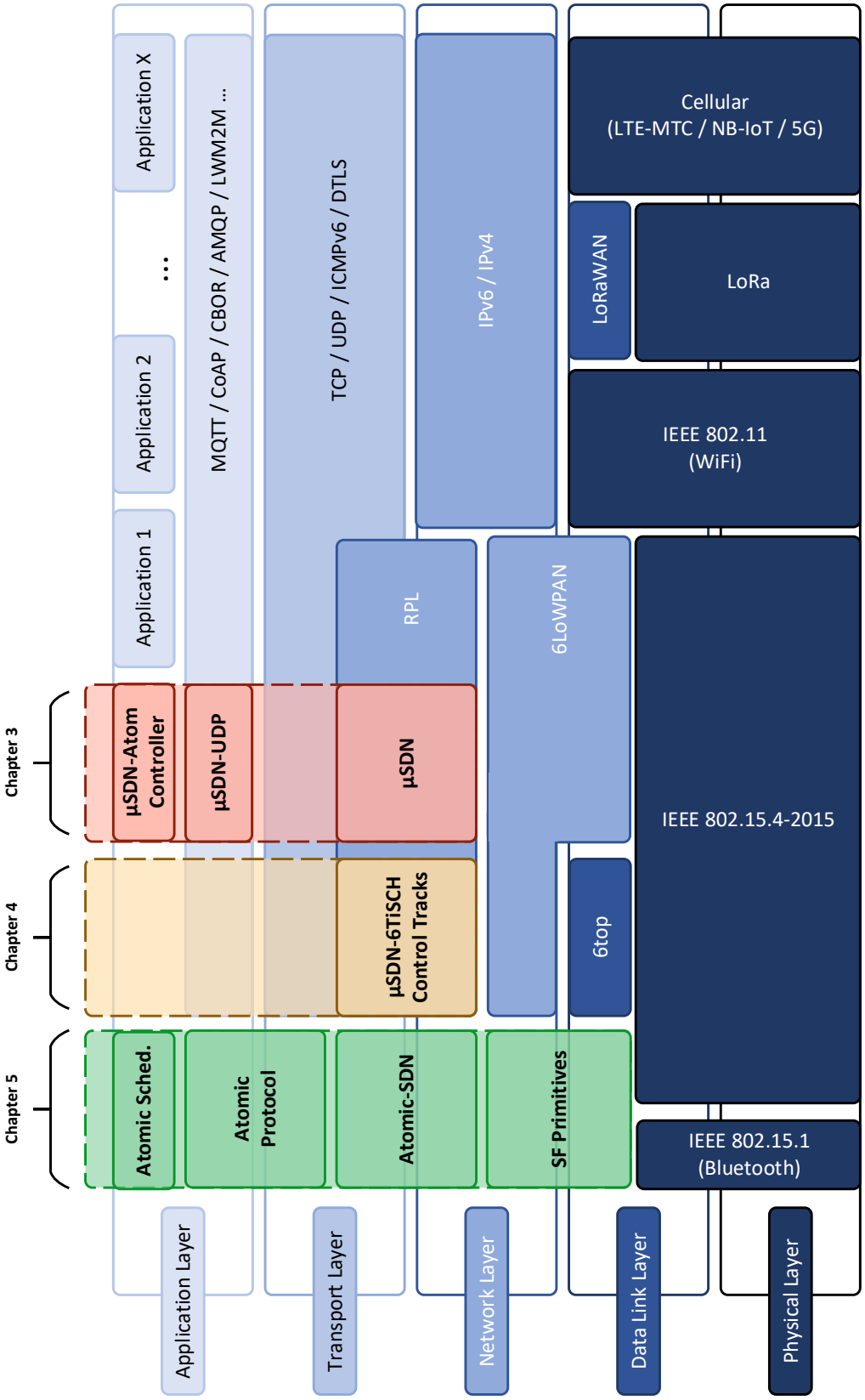


Figure 1.3: Thesis scope with respect to IoT protocols.

CHAPTER 2

Background Material and Relevant Literature

Over the past decade Software Defined Networking (SDN) has been a significant area of networks research. Chapter 1 introduced how initial strides have seen SDN being used to overhaul the networking architecture of major businesses, and how there is a keen interest in leveraging this research to solve the challenges faced in current and future IoT networks. Ever more sophisticated devices and new low-power wireless technologies are driving how IoT networks can be deployed, and the benefits that they bring. Industrial IoT (IIoT) networks, with varied and unique application requirements, will play an important role in this conversation.

The increased flexibility and fine-grained control made available through the adoption of SDN concepts, promise more intelligent use of resources, redistribution of this intelligence within the network, and complements the lightweight and energy efficient protocols (such as IEEE 802.15.4, 6LoWPAN and RPL) that govern low-power wireless networks. Decisions regarding what data to forward or else discard, and the management of individual nodes, can be taken at controller level with a global view: the implications of which directly affect reliability, delay, and power consumption within the network

This chapter covers both necessary background information on a number of topics introduced in this thesis, and provides a comprehensive literature review on SDN within low-power wireless environments. The following summarises the contents of this chapter:

- An introduction of background material examining SDN in wired networks, showing how it provides a platform for key concepts such as Network Virtualisation, Network Function Virtualisation, and Network Slicing.
- An overview of low-power mesh networks and the IEEE 802.15.4 'IoT Stack', showing how its various layers have extended IPv6-enabled Internet of Things to even the most

constrained devices.

- An introduction to 6TiSCH Industrial IoT and how best-practice solutions from other industrial wireless protocols, as well as concepts from SDN architecture, have been incorporated into standardisation efforts from the IETF 6TiSCH Working Group .
- Relevant background information and literature on CT and SF in low-power wireless networks; an understanding of which is crucial for the technical content covered in Chapter 5.
- A comprehensive review of recent research examining SDN in IoT, with particular focus on low-power wireless mesh networks.

2.1 Software Defined Networking and the 'Network Operating System'

To provide the reader with a greater understanding and background to the technical work presented in this thesis, particularly the μSDN and Atomic-SDN architectures introduced in Chapters 3 and 5, this section explores the concept of Software Defined Networking (SDN) and outlines its main architectural components. However, it is important to note that although this section provides examples of key literature on SDN in wired networks, this particular research area is not the main focus of this thesis. For a detailed analysis and listing of protocols, controllers, and applications, the reader should be directed toward a number of key SDN surveys: [4, 19, 30, 31].

2.1.1 General Approach

The operation of computer networks can be divided across three succinct planes: *data*, *control*, and *management*. The *data plane* handles the passing of information between network devices, the network *control plane* defines the functions and processes that dictate how this information is exchanged across the whole network, while the *management plane* consists of the higher-level policies that define the desired network response.

In conventional wired networks these three planes were tightly coupled within vertically integrated architectures. Network services such as load balancers, firewalls, and routers were provided through vendor-specific (and often costly) hardware, and network configuration changes often had to be made manually, across multiple devices. The complexity resulting from this approach meant that not only were these changes time-consuming, driving up Operational Expenditure (OPEX) costs, but potential capacity within the network needed to be accounted and provisioned for from the outset, further driving up Capital Expenditure (CAPEX).

In contrast, Software Defined Networking (SDN) advocates horizontal separation between data forwarding devices and the control plane. Routers and switches are relegated to 'dumb' forwarding devices, and control decisions are made on a logically centralised control plane.

Figure 1.2 in the previous Chapter illustrates this idea: of how SDN has shifted the view of networks as vertical systems (i.e. with each device performing a specific task, often with additional common functionality layered on top of it) to one of a horizontally distributed control architecture. Instead of the control plane (network traffic decisions) and data plane (the forwarding of traffic based on control plane decisions) residing on the same device, the control plane is abstracted to one or more dedicated SDN controllers. Following a mix of academic research and industry efforts, at its heart SDN brings decades of best-practice from computer architecture and imbues a measure of programmability to network infrastructure, and there has been considerable progress in both standardisation and commercial acceptance.

The advantages of moving to this model are twofold. Firstly, knowledge of the network state is now logically centralised across the control plane (conversely, this can often be physically decentralised across multiple distributed SDN controllers). This centralisation of intelligence allows controllers to make informed decisions with respect to a global view of the current network state, rather than applications making decisions based on local knowledge.

Secondly, the particularities and fine-grained management of network resources are now abstracted away from high-level functionality, allowing network applications to be configured and instantiated without knowledge of the underlying hardware. Like the Operating System (OS) on a computer, where programmers rarely need to concern themselves with the allocation of memory and Central Processing Unit (CPU) resources, management of the low-level hardware is contracted to other processes. This has given rise to the Network Operating System (NOS) moniker, which sees SDN as an enabler for viewing the network as a single distributed system, on which virtual network applications can be 'programmed' (though the reader should note that the virtualisation of network functions, commonly referred to as NFV, is distinct from the programmable framework enabled by the NOS).

This concept, that of SDN as an architectural enabler for a NOS, is presented in Figure 2.1, which expands upon the SDN Layer Architecture as introduced by Internet Research Task Force (IRTF) RFC 7426 [5] and attempts to define what constitutes the various components and architectural structure of SDN. Although this figure departs marginally from RFC 7426 model, it endeavours to extend the architecture with ideas and concepts from key literature, and authors who have been influential within the SDN research space [4, 19, 30–32]. Crucially, this model defines the SDN control plane as consisting of three key abstractions:

1. A Network State Abstraction Layer (NSAL) allows applications to interface with the distributed network state, providing simplified (virtual) network models and translating required network behaviours into control operations.

2. A (distributed) data store and control plane separates physical infrastructure from the network applications, providing a global network view and shielding applications from the vagaries of the physical network state.

3. A Device Abstraction Layer (DAL) provides a flexible forwarding model and abstraction, hiding the details of the underlying hardware and allowing SDN controllers to communicate with and configure devices in a platform-agnostic fashion.

Together these three abstractions divide the complexity of network management into tractable pieces. The remainder of this section further explores Figure 2.1, following a bottom-up approach in terms of the traditional *Data*, *Control*, and *Management* planes. Within each layer it tries to define the abstractions that support the distinctions between the SDN layers, as well as explore associated concepts that are inexorably linked with SDN, and which have been key to the success of SDN as a 'platform for programmability'.

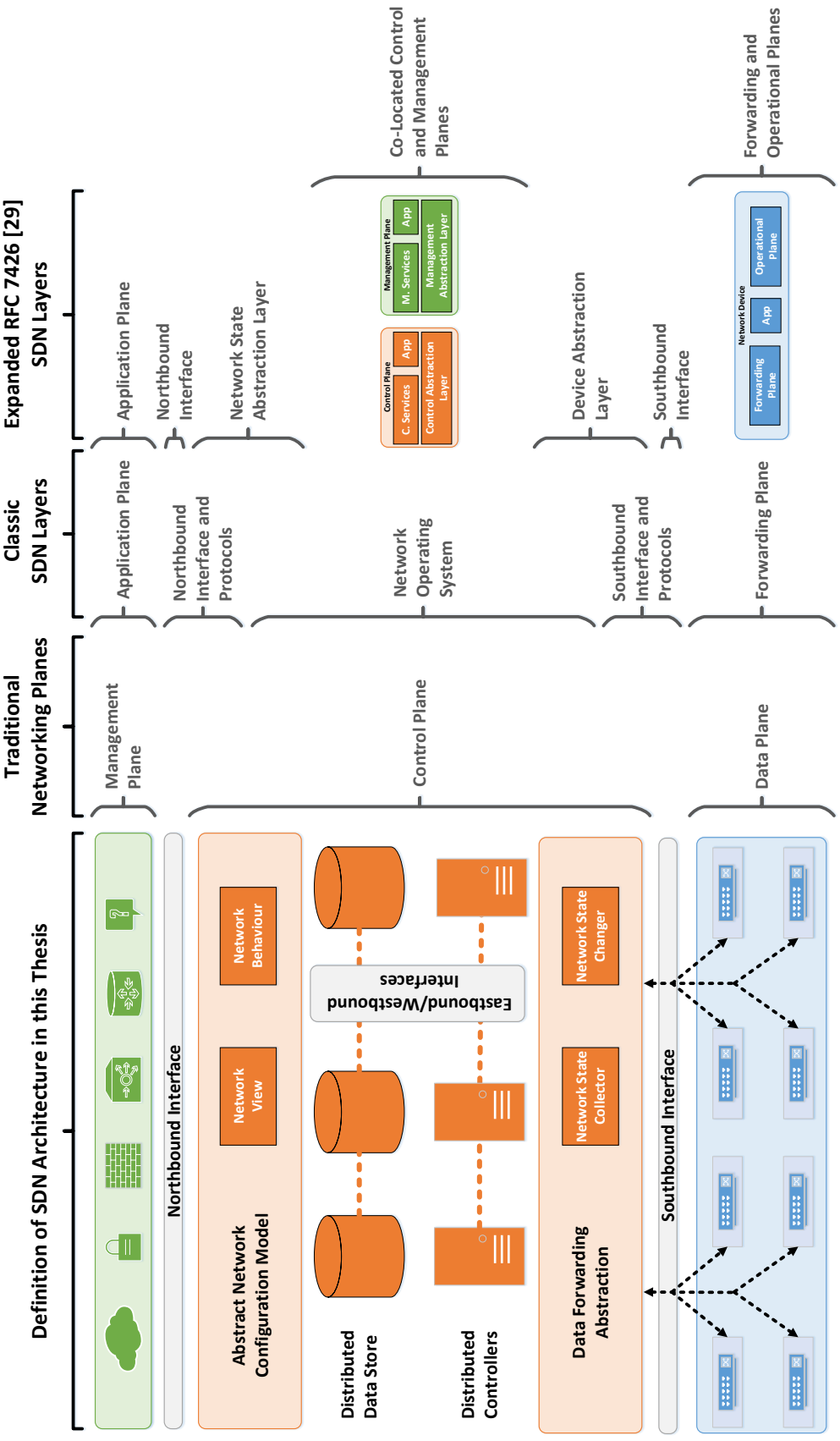


Figure 2.1: SDN architecture as defined within this thesis. With reference to the view of traditional networking planes and classic SDN layers as outlined in [4], and the expanded interpretation of SDN layers discussed RFC 7426 [5].

2.1.2 Data Plane

As previously discussed, the core idea behind SDN lies in the separation of data forwarding processes from network control mechanisms. Within the overall SDN architecture, this results in physical network devices on which control complexities are abstracted to higher layers; what remains are simple forwarding devices, incapable of autonomous decision making, and devoid of intelligence. Following the computer operating system analogy from earlier, this can be thought of as the computing resources available to the programmer. For these resources to be of any use, standard and well-defined interfaces must be defined that allows these blank resources to be configured in a manner that results in the desired functionality: the most notable of which has been OpenFlow [33, 34].

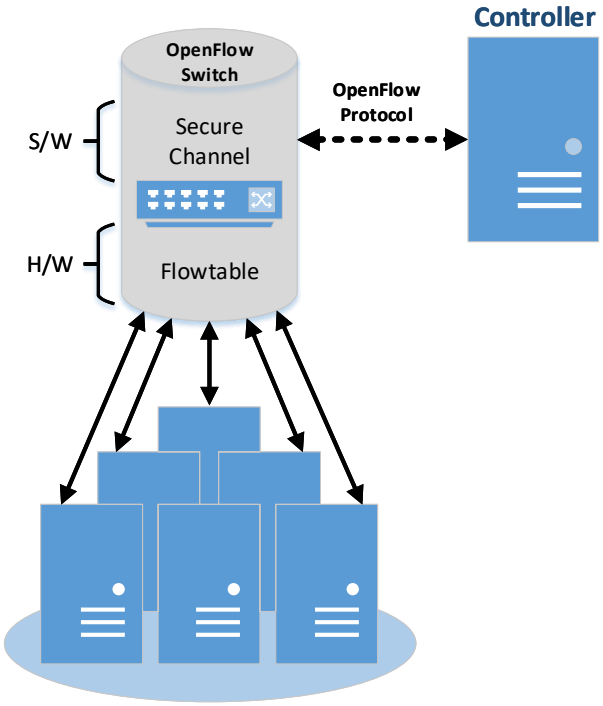


Figure 2.2: OpenFlow architectural concept.

Understanding the motivation behind the OpenFlow protocol is key to understanding the abstracted SDN data plane (although the reader should note that it is not the only available southbound SDN protocol, which are expanded upon in the proceeding section). Originally proposed in 2008 as a solution to make campus networks more manageable, OpenFlow aimed to exploit common functions within proprietary switches and routers in order to provide a secure and open-source protocol allowing devices to be programmed agnostically via a centralised controller, as shown in Figure 2.2. Specifically, OpenFlow provides a standardised language to allow SDN controllers to alter the state of the network through the manipulation of device flowtables: where a flow is defined as a set of data packets with common properties (for example, common header

fields) passing through a nexus within the network; and a flowtable is defined as a table of per-flow forwarding rules governing ingress and egress traffic.

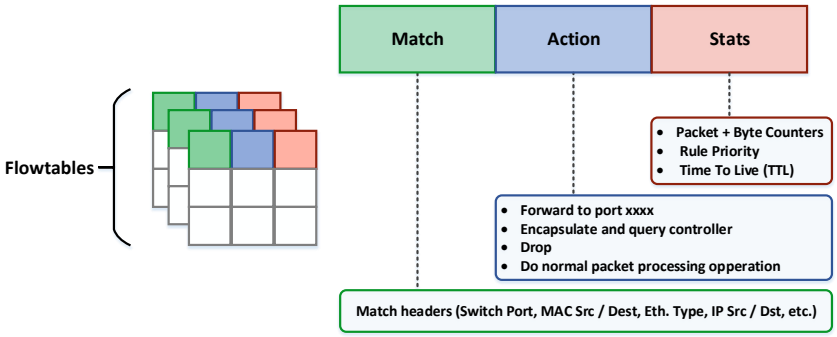


Figure 2.3: OpenFlow flowtable structure.

SDN flowtables, as opposed to standard routing tables, are the key data plane mechanism in moving from traditional network switches to a programmable SDN architecture. As summarised in Figure 2.3, flowtables determine how packets are matched and what action should be taken (i.e. whether they should be forwarded or dropped). Additionally, flowtables maintain statistics of each data flow, reporting this data to the controller and allowing them to make decisions based on the current network status. Although each rule is a simple instruction, by passing ingress packets through a chain of multiple flowtable rules complex functionality can be created. Consequently OpenFlow allows switches to be programmed to act as a myriad of devices, such as routers, switches, firewalls, or load balancers, from an abstracted control plane operating elsewhere within the network.

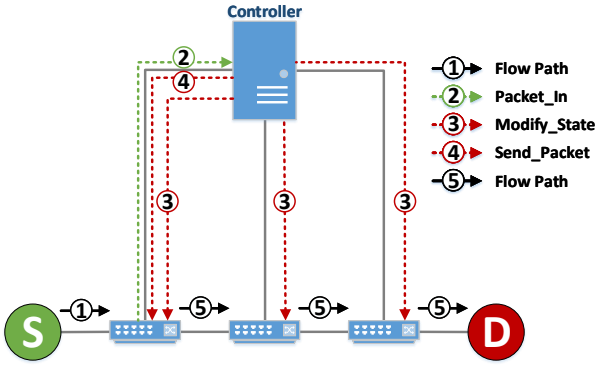


Figure 2.4: OpenFlow routing example. The messages referenced within the figure are defined as part of the OpenFlow protocol [33].

The routing example presented in Figure 2.4 shows how an OpenFlow data forwarding device handles a new flow within the network. In this example, a source node S is trying to route a packet to a destination node D. As none of the intermediate switches have knowledge of how

to handle this flow, the first switch both buffers that packet (so that it may later send it), and encapsulates it within a OpenFlow $Packet\_In$ message to the controller. This message is received by the controller, which makes a decision on how to route the flow $S \rightarrow D$. Each switch along the path is then sent a $Modify\_State$ message, which instructs it on how to handle the new flow by installing rules on the flowtable. Once each switch along the route is configured, the controller notifies the original switch through a $Packet\_Send$ message that it is now safe to send the original packet. As each switch along the route has been configured to handle the new $FlowPath$, the packet is successfully received at D.

2.1.3 Control Plane

The SDN control plane is the central pillar of SDN architecture, describing a multi-faceted concept where a number of key components are linked together to provide logically centralised control and network configuration: the *Southbound (SB) Interface* and associated protocols, a *SDN Controller* or *Network Operating System (NOS)*, and the *Northbound (NB) Interface*. This subsection summarises these architectural layers for the reader, and highlights how research and industry standardisation efforts have helped shape SDN control architecture.

2.1.3.1 Southbound Interface

Within computer networks Southbound (SB) Interfaces are a simple concept separating control elements (both hardware and software) from the data forwarding devices. In traditional networks, this interface sits on the individual devices and tightly integrates the data and control planes. However, within the context of SDN, this interface is used to decouple them, through a SB communications protocol. The SB interface provides a formal definition of interaction, allowing the manipulation of data flows by high performance controllers placed elsewhere in the network. The SB protocol encapsulates the fundamental principle of SDN: abstraction between the forwarding and control planes. Well-defined SB SDN protocols such as OpenFlow, described in detail in the previous section, therefore facilitate communication across this interface, and provide a means of elevating network functions to the controller.

However, although OpenFlow was the original 'SDN protocol', and still remains highly popular amongst industrial vendors, it is not the only southbound solution. Not only have a number of different SB protocols been proposed and implemented since then; such as Protocol Oblivious Forwarding (POF) [35], Hardware Abstraction Layer (HAL) [36], Open vSwitch Database (OVSDB) [37], OpFlex [38], and OpenState [39]; but concepts such as programmable networks and active networks [40, 41], including forwarding abstraction protocols such as Forwarding and Control Element Separation (ForCES) [42], define some key SDN concepts while predating the original OpenFlow paper [33].

Rather than proposing a set of new instructions, OpenState [39] instead extends the concept of match/action flowtables within OpenFlow, supporting mechanisms to implement Finite State

Machines (FSMs) within the switches. This allows stateful logic to be run within the forwarding devices themselves, without increasing the complexity or communication overhead of the control plane, and allows decision making that requires only local knowledge to be configured programmatically onto the device itself.

OpFlex [38] is a hybrid SDN protocol that distributes some of the complexity back to the forwarding devices in order to increase network scalability. Whereas a 'pure' OpenFlow definition would insist on zero intelligence within the data forwarding devices, OpFlex takes a more elastic view by centrally defining an overall control policy, before distributing this to data path elements that are able to use the policy to make some local decisions.

OVSDB [37] is complementary to OpenFlow. As virtualisation techniques are easily supported through SDN architecture, virtual switches such as Open vSwitch (OVS) [43] have grown in popularity to become an integral component of SDN based networks. OVSDB provides management extensions for OVS, supporting additional network capabilities such as the instantiation and tearing down of virtual devices, and modifying OVS bridges, ports, and interfaces.

However, southbound SDN protocols have not completely replaced legacy device configuration protocols. Particularly when SDN was first adopted by industry, there was interest in retaining interoperability and defining southbound plug-ins for protocols such as Border Gateway Protocol (BGP) [44], Simple Network Management Protocol (SNMP) [45], Network Configuration Protocol (NETCONF) [46], Path Computation Element Communication (PCEP) [47], and Extensible Messaging and Presence Protocol (XMPP) [48]. Efforts such as HAL [36] attempted to address this by defining an additional abstraction layer that sits below the SB, in order to transform legacy network elements into OpenFlow capable devices fully interoperable with higher SDN layers.

Yet these examples continue to suffer from one of the key weaknesses of OpenFlow: the initial design choice to match on static header fields. This has resulted in continuous revisions to the original specification, as the Open Networking Foundation (ONF) governing body seeks to extend OpenFlow capabilities to a number of different network and traffic types. Each revision expands the number of compatible header fields, such as IPv6, Multiprotocol Switch Labeling (MPLS) [49], and Virtual Extensible LAN (VXLAN) [50]. This, alongside backward compatibility requirements, has therefore gradually increased complexity; an anathema to the original goals of SDN.

To address this weakness, POF [35] provides a means of abstracting the way that flows are matched within the flowtable. Rather than matching on specific header fields, as in OpenFlow, matches are instead performed on index/length pairs that are defined in configuration messages from the controller. This allows switches to process data flows without knowledge of the underlying protocol.

Similarly, P4 [51, 52] also provides protocol independent forwarding. However, whereas POF allows the operator to define what the underlying primitive instruction set should be, P4 extends this by allowing the operator to define how packet processing programs can be written and

compiled, using the POF instruction set. In essence, P4 goes further than other approaches in that it is not a protocol but a programming language in its own right; like C, JavaScript, or any other computer programming language. It provides a target-independent way of configuring the network data plane. Network behaviour is defined in a P4 script, which is then compiled into a forwarding plane configuration by a P4 parser on the switch. This differs from OpenFlow in that while OpenFlow is an instruction set to allow forwarding elements to communicate with the control plane, P4 is capable of defining *how* this instruction set might look. Indeed, it is possible to program P4 switches to behave as OpenFlow capable switches.

2.1.3.2 SDN Controller / Network Operating System (NOS)

As discussed in Chapter 1, a useful analogy for SDN is that of a computer operating system. SDN promotes the idea that, as computer programmers shouldn't have to concern themselves with managing access to low level computing resources, neither should network operators have to concern themselves with the intricacies of managing the network data plane.

In both industrial and academic communities the NOS is commonly referred to as the SDN controller, although these terms can be used interchangeably. In contrast to previous approaches, the standard SDN model places the NOS so that it is logically centralised within the architecture. Referencing the classic SDN layers in Figure 2.1, the abstractions that SDN provides at the Northbound (NB) and SB interfaces expose standard APIs to network functions on the Application Plane, as well as the data forwarding elements. Additionally, the abstract nature of these APIs mean that although the SDN controller is logically centralised, there is no need to mirror this physically. They can be viewed as a single unified interface, and many modern SDN controller implementations manage control distribution across the network, defining Eastbound/Westbound interfaces for distributed control protocols.

The SDN controller, or NOS, can be considered in three layers: the data plane orientated *Device Abstraction Layer (DAL)*, a distributed state managed through *Eastbound / Westbound Interfaces*, and the application-facing *Network State Abstraction Layer (NSAL)*:

Device Abstraction Layer (DAL): While the SB interface defines the protocols and interactions when communicating between the control plane and the data forwarding elements, the DAL abstracts the resources present on those devices, based on models held in the SDN controller. This could be a Hardware Abstraction Layer (HAL) [36], as described earlier, however it does not necessarily need to refer to a physical device. Not only may it abstract the resources of virtual devices (such as those managed by a hypervisor [53]), but it presents a unified view of all data plane resources (both forwarding and operational) to the SDN controller. Providing functions to receive information from, and send configuration to, all devices, no matter the underlying physical/virtual differences.

Eastbound/Westbound Interfaces: Although initial efforts in SDN research originally considered the network as managed by a single physical controller, key research questions con-

cerning scalability within industrial scenarios promoted the idea that the SDN control plane could be thought of as logically centralised, but physically distributed. In large SDN deployments there can often be a number of distributed controllers operating across multiple physical networks. In such a network, the SDN controller implementation defines Eastbound/Westbound interfaces to allow the distribution of intelligence amongst multiple control instances, or even prior-generation control systems. For example, this could include the exchange of topology information between controllers, in order to coordinate the flow of network traffic, or integration with legacy control plane services, such as load balance servers. In order to provide these services, Eastbound/Westbound interfaces require distributed data management and queuing protocols (such as AMQP [54]) in order to deal with issues of concurrency. Although this adds complexity to the distributed SDN architecture, the potential benefits include being able to increase the robustness of the system through diversity, increased interoperability of heterogeneous devices and the potential for greater data processing and computing power.

Network State Abstraction Layer (NSAL): Sitting below the NB interface the NSAL provides similar functions to that of the DAL, servicing messages to/from the application layer and providing a unified Abstract Programming Interface (API) to the SDN control plane. The NSAL provides abstracted network views to applications, whilst interpreting network policies from those applications and passing them to the appropriate functional blocks within the SDN controller. For example, an application may need to re-route traffic across the network. If it considers a virtual view of the network topology, it has no knowledge of how to achieve this outcome at the data plane. By providing the NSAL with a general network *policy* of what it wants to achieve, the application delegates to the controller, which can then enact this policy using the appropriate control functions.

2.1.3.3 Northbound Interface

Residing at the top of the SDN control plane is the Northbound (NB) interface, which provides a standard mapping between the northbound functions of the NSAL and application layer services. Revisiting the computer OS analogy, the NB interface can be compared to IEEE POSIX [55]: which defines a standard shell interface to abstract higher level applications away from the OS. However, while OpenFlow has established itself as the de facto southbound SDN protocol, stakeholders have been slow to coalesce around a common northbound counterpart, and approaches range from RESTful interfaces [56], to intent-based solutions [57].

2.1.4 Management Plane

In computer networks the goal of the management plane is to define network policies which can then be translated into control instructions that are enacted by data forwarding elements. Within traditional IP networks, where the planes are tightly coupled on each device, these policies

are typically configured onto each network device in a decentralised fashion through multiple Command Line Interfaces (CLIs). A laborious and costly process.

SDN divorces network management applications (such as routers, firewalls, and monitoring) from the physical devices. In the classic understanding of SDN architecture, the idea of a management plane is blurred with that of the control plane, and these two terms were often used interchangeably. However, as depicted in Figure 2.1, RFC 7476 [5] attempts to extend the layered SDN model by placing the management plane alongside the control plane, with a high-level application layer sitting above the NB interface. In this understanding, the management plane is taken to mean the set of applications that utilise the control functionality provided by the NB interface in order to dictate network policy. Using the instruction set provided by NB protocols, these policies are interpreted by the SDN controller and enacted as specific forwarding behaviours at the data plane.

This decision was taken based on the understanding that control tends to consider only data forwarding, while management is mostly related to the operational aspects of the network, such as the management of device resources. The application plane is still where high-level network policy is defined, however control functionality that directly affects the operation of data forwarding elements is delegated to the control and management planes. This more nuanced view, of co-locating the two SDN architectural layers has been adopted by modern SDN controllers such as OpenDaylight (ODL) [58] and Open Networking Operating System (ONOS) [59], and is inherent in network programming languages like P4 [52].

2.1.5 Virtualisation

The authors of [60] introduce a test to help with the distinguishing between abstraction and virtualisation: defining virtualisation as a process where the input units are the same as the output units, thus allowing *recursive* virtual layers; if this recursivity test cannot be met, then it is not considered virtualisation. They differentiate this with abstraction in that the latter solely hides complexity, and can not be layered recursively. Given that SDN and virtualisation are often two deeply intertwined concepts, it is important to understand this distinction.

Software Defined Networking provides abstraction layers to programmatically configure control applications onto data forwarding elements, while not having to worry about the finer details of how this happens. Network virtualisation, on the other-hand, embodies the idea that the physical network resources can be sliced into smaller chunks, and presented to higher level processes as if they were those same resources. Multiple virtual elements can then exist on a single physical entity or stacked over multiple layers. SDN-based virtualisation typically falls into three categories: *device virtualisation*, *network slicing/hypervisors*, and *network function virtualisation*. These three aspects are mapped as part of the recently completed IRTF Network Function Virtualisation Research Group (NFVRG) [61], which provides reference architecture, details the interlinking between NFV and SDN, and explores virtualisation research challenges.

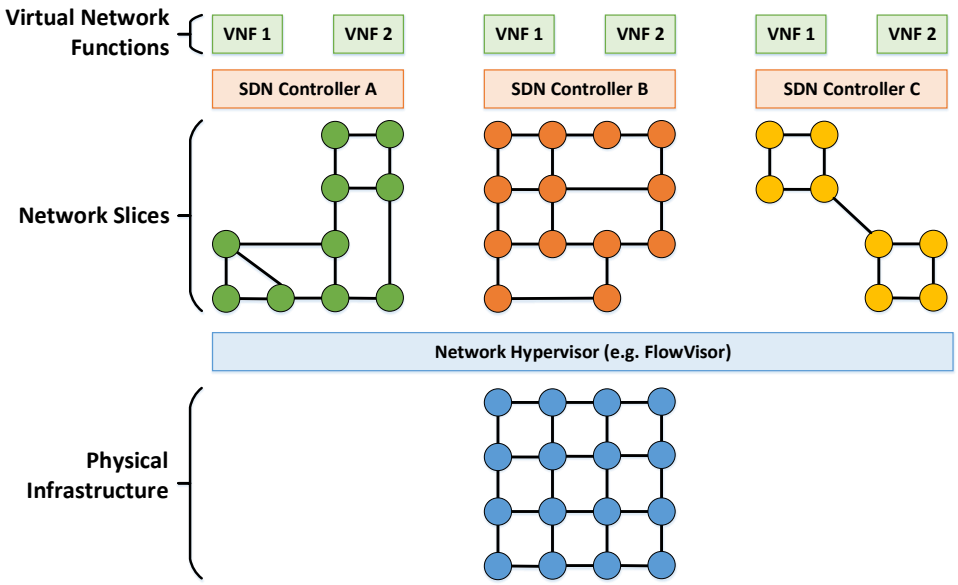


Figure 2.5: Multiple logical devices and networks, sliced from a single physical topology.

1) Device Virtualisation: The majority of vendors now support OpenFlow APIs in their switches, and OpenFlow flowtables provide a powerful tool that allow fine-grained and reactive manipulation of data flows. The powerful hardware specification typical in these devices means SDN switches are capable of storing thousands of flowtable entries and handling hundreds of control messages per second. However the support enjoyed by SDN, and in particular OpenFlow, stems directly from the framing of SDN as a platform for virtualisation. Network elements don't necessarily need to be hardware devices, and software switches such as Open vSwitch [43], allow network operators to create virtual data forwarding elements.

2) Network Slicing / Hypervisors: One of the original attempts to incorporate virtualisation into the SDN architecture was FlowVisor [62], which extends the concept of hypervisors (commonly used to virtualise computing resources for cloud services) to the network domain [53]. FlowVisor uses network slicing, whereby physical resources are 'sliced' amongst multiple virtual instances to create logical views of the physical network. It essentially acts as a proxy between the individual controllers attributed to each slice, and intercepts all messages between data forwarding elements and their SDN controllers. This allows any single physical OpenFlow switch to be turned into multiple virtual switches. Each slice is supported by a single active controller, which can only manage the traffic within its own particular slice, and only within the bounds of the QoS provisions placed on that slice by the network operator. The FlowVisor approach the network slicing concept to five primary slicing dimensions: *bandwidth*, *topology*, *traffic*, *device CPU*, and *forwarding tables*. These resources can then be shared out over network slices as a portion of the total available network resource, providing multiple logical network topologies as shown in Figure 2.5. Different services can then be attributed to different logical networks, which

in turn have different QoS requirements.

3) Network Function Virtualisation: Both NFV and SDN share the goal of introducing greater programmability into network architecture, eschewing dedicated hardware systems for generic and configurable software based control. This similarity has (incorrectly) lead to the two concepts being used interchangeably. NFV can be achieved without SDN, and vice versa, however they are complementary ideas that benefit from one another. Whereas SDN deals with the separation of the data forwarding from the control systems, NFV focuses on leveraging virtualisation techniques (such as network slicing, and the control abstractions provided by SDN) to provide virtual network infrastructure to services, which can then be decomposed into a set of Virtual Network Functions (VNFs) [63]. By elevating applications to the virtual plane, rather than tying them to the physical infrastructure, network functionality can be more easily managed and provisioned: with multiple virtual applications used in sequence to provide a particular service.

2.1.6 Applications and Use-Cases

The combination of SDN, coupled with NFV, provides network operators with an extensive and powerful toolbox capable of addressing some of the key issues inherent in traditional networks. Not only does SDN's global network view and configurability have significant ramifications in network security, but the virtualisation capabilities of SDN hypervisors combined with NFV are being used to provide scalable and elastic network services. These applications and others, are summarised in Table 2.1.

2.1.7 Conclusions

From this overview of SDN architecture and relevant literature, a number of conclusions are drawn with reference to Research Questions [Q1, Q3], which form the basis for much of the technical content presented in Chapters 3 and 4:

[Q1]: *What are the fundamental features of a minimal SDN solution?* This subsection reviewed the layered abstractions and interfaces of SDN architecture. From this overview it is clear that the original definition of SDN has evolved, as the strict understanding of SDN based on the OpenFlow model came up against legacy requirements from vendors, and an increasing need for elastic deployment of intelligence in the network. Given the examples and interpretations covered, this thesis considers the following as a minimal layered SDN architecture.

- Applications that manipulate network state.
- Northbound abstractions that present and configure network state.
- A logical data store of the abstracted network state.
- A configurable mechanism to manipulate network resources.

Table 2.1: Summary of SDN applications.

Application	Examples
Traffic Engineering (network planning and monitoring)	Energy-aware network utilisation
	Maximising network utilisation
	Optimised load balancing
	Traffic optimisation techniques
Mobility and Wireless (optimisation of the following)	Sharing limited spectrum
	Allocating radio resources
	Implementing handover mechanisms
	Interference management
	Load balancing between cells
Measurement and Monitoring	Providing new services and information
	Improving SDN protocol performance
	Monitoring cloud infrastructure
Security and Resilience	Replacing middle boxes
	Detect DDoS attacks
	Detect IP spoofing
	Deep packet inspection

- Southbound abstractions to present and configure these resources.

Additionally, the OpenFlow flowtable model [33] identifies three situations in which the control plane will interact with the data plane. Following this model, any interpretation of SDN for low-power wireless must, at a minimum, provide these services.

1. *Collection* services that allow the SDN control plane to monitor the global network state through flow statistics.
2. *Configuration* (unsolicited) of the network triggered by the controller itself.
3. *Reaction* events triggered by unknown flows, where a device needs to query the controller in order to receive instruction on how to handle the flow.

[Q3]: *Are there solutions to reduce SDN overhead?* The control overhead generated by SDN protocols at the SB interface, i.e. the communication messages sent between the data and control

planes, is an integral part of any SDN architecture. However, it can present challenges even in wired networks with powerful SDN switches and a number of the southbound SDN protocol solutions described in Subsection 2.1.3.1 attempt to address this problem.

- OpenState [39] returns control to the data forwarding elements by implementing state machines in the OpenFlow flowtable
- OpFlex [38] additionally recognises that excessive control communication can impact network scalability, and returns some control to the forwarding elements.
- POF [35] and P4 [51] provide target and protocol independence, addressing compatibility and upgrade issues between devices.

2.2 Low-Power Wireless for Industrial IoT

Although the background information provided in this section is not an exhaustive analysis of low-power wireless protocols, it is intended to provide the reader with an understanding of the challenges and limitations faced by the constrained mesh networks commonly deployed in Industrial IoT (IIoT) networks and, in particular, the difficulties of implementing centralised control mechanisms within such an environment. These challenges form the context and motivation for the technical content (Chapters 3 to 5) of this thesis. To this end, this section covers three distinct areas of low-power wireless networks: firstly *an introduction to wireless mesh networks*, outlining the unique challenges imposed by a wireless mesh topology; secondly, *an overview of specifications and protocols* that, together, make up the IEEE 802.15.4 low-power 'IoT Stack'; and finally, *a summary of IETF 6TiSCH* [2], showing how SDN concepts were incorporated into the standard.

2.2.1 Wireless Mesh Networks

Multi-hop mesh networking describes a concept whereby devices are able to form ad-hoc links with neighbouring nodes, creating a series of connections where all nodes are able to communicate with one another by navigating these links (or 'hops').

Mesh networking allows devices to extend their range outside that of what their physical layer would normally permit, with other network nodes forwarding messages to the intended destination. An example of this is shown in Figure 2.6, where node S is able to send a message to node D via the route $S \rightarrow 7 \rightarrow 5 \rightarrow 6 \rightarrow D$. This message is then passed on to a backbone network, facilitated through the border router operating at D. Furthermore, the spatial diversity inherent in a mesh topology allows better utilisation of the limited radio spectrum resources. Assuming single channel contention at the MAC layer, spatially separated transmitting (T_x) and receiving (R_x) node pairs can communicate concurrently without interfering (though this of

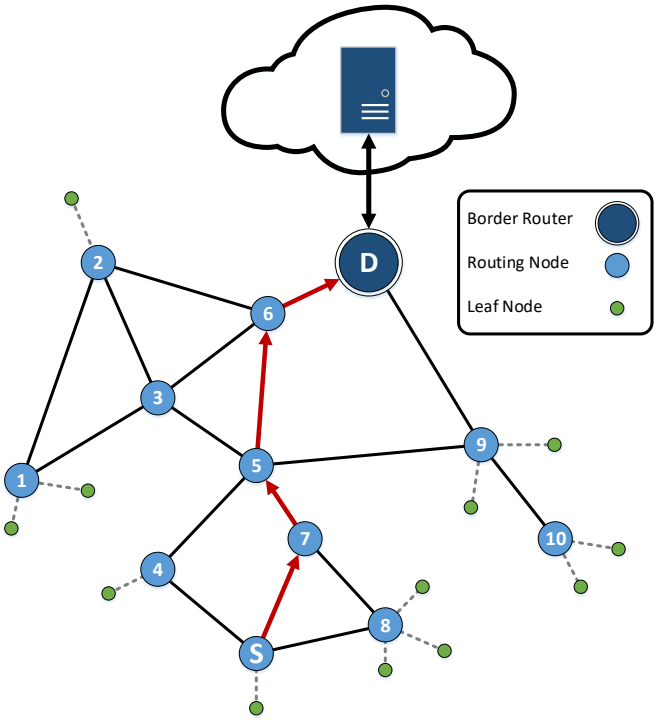


Figure 2.6: Low-Power mesh network with border router.

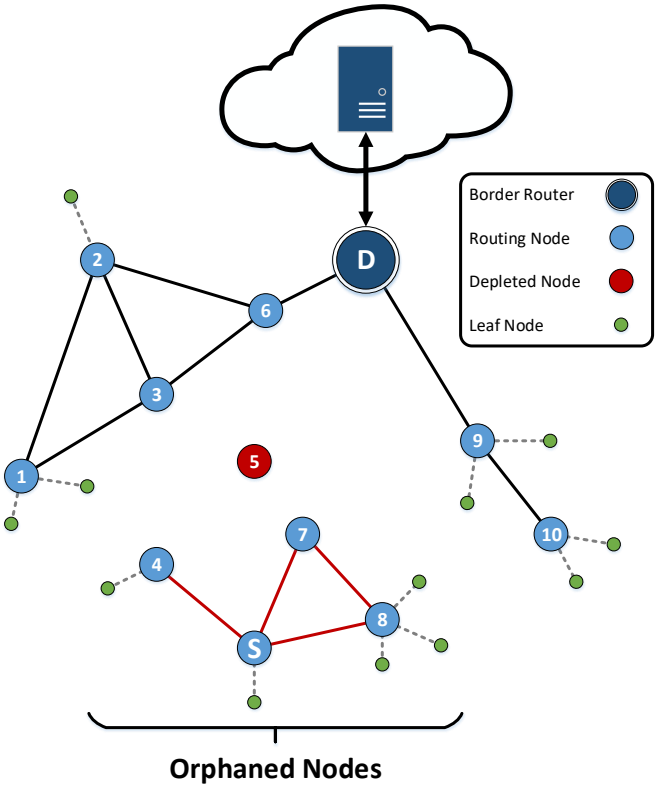


Figure 2.7: Low-power mesh with orphaned branch.

course depends on the range of the underlying physical layer options). However, mesh solutions also add a considerable complexity to communications, arising from some key characteristics: *number and density of nodes*, *traffic characteristics*, and *power* or *duty-cycling* requirements.

2.2.1.1 Network Size and Density

Both the total number and density of nodes have a profound impact on mesh network performance. Alleviated or exacerbated by the choice of routing and MAC protocols, which govern how and when a node can communicate, the scale and layout of the network add a considerable layer of complexity to the mesh.

As discussed, mesh networks can provide a low-cost, low-power infrastructure solutions for IoT. In industrial scenarios, this can often mean covering large areas with hundreds or even thousands of nodes for a single mesh network [14]. Not only can this result in large hop distances of 10 to 20 nodes, further exacerbating the issue of how to route packets and introducing multiple points of failure in an end-to-end link, but dense clusters of nodes located in close proximity can cause self-interference within the network. This is a particularly acute problem in the Carrier-Sense Multiple Access / Collision Avoidance (CSMA/CA) MAC layer traditionally employed in IEEE 802.15.4, which sits on a single channel without employing frequency hopping. Co-located nodes consequently suffer from the *hidden node problem* commonly experienced in wireless communications, whereby two nodes that have no knowledge of each other try to transmit to a third node at the same time, resulting in a collision (though, as covered later in this Chapter, if the nodes are sufficiently synchronised then the receiving node can actually demodulate the signal given certain physical layers [64, 65]).

2.2.1.2 Topology and Traffic

The Internet of Things (IoT) is a catch-all term that encompasses a vast array of devices, technologies, and applications. Although traditional IoT applications have focused on data collection scenarios provided by WSNs, there is no single overarching communications pattern that IoT networks need to address. Indeed, although this thesis focuses on mesh networking, there is nothing to say that an IoT deployment can't adhere to other topologies, as outlined in the IEEE 802.15.4 standard [1]. Additionally, data traffic might be low-rate, or it might be bursty; there may be multiple applications or multiple services on a single network; and the network may be designed for *many-to-one* upwards communications, as in WSNs, or it may be a *one-to-many* actuator network.

However, the prevailing choice for WSNs has been towards self-organising mesh networks that funnel data toward a sink node. This has been driven by the success of the RPL routing protocol [24], examined more extensively in Section 2.2.2, which provides lightweight means of establishing ad-hoc *many-to-one* data collection in a low-power and lossy network. This distributed approach to mesh topology construction reduces signalling overhead in comparison to

centralised methods, and ensures that link failures don't break the topology. However, with limited information on the health of the overall network, nodes need to rely on their local knowledge in order to choose links, resulting in graphs that may be poorly optimised for the current network state. The argument between centralised and distributed approaches to constructing a mesh topology is therefore one of optimisation based on global knowledge, versus a leaner distributed network that is better able to scale.

Yet focusing solely on topology runs the risk of simplifying the problem, as topology is closely intertwined with traffic characteristics. The tree-like graphs created by RPL are ideal for data collection. However, for many IoT scenarios, there is a need to support not only *many-to-one* data collection protocols but also downwards *one-to-many* communications from the sink node. This can emerge as a requirement from actuator networks, delivering over-the-air firmware updates to devices, or even to support reliable upper-layer protocols where Acknowledgement (ACK) based protocols need the reverse *one-to-many* path established: such as TCP, or CoAP confirmable requests [66]. How best to build a routing topology that can service multiple traffic patterns, applications, and data rates, becomes a complicated trade-off where favouring one application scenario may adversely affect performance for another.

2.2.1.3 Power and Duty-Cycling

The use of low-power protocols in wireless mesh networks is often driven by the energy restrictions of battery-powered devices, which can be common in application scenarios where nodes may be difficult to reach, or there is no power source available. Although there has been considerable research into energy harvesting IoT devices [67, 68], there is still a danger that excessive communications across a single node, whether through application traffic characteristics or by virtue of the topology, can cause branches within the mesh to become orphaned from the wider network. An example of this is shown in Figure 2.7, where node 5 has been depleted and can no longer serve as a bridging node to the source node S from the previous example, along with nodes 4, 7, and 8. In such a case, the topology construction protocol needs to dictate how these nodes are able to rejoin. If these nodes are not in range of another (connected) node, then that branch is orphaned from the rest of the network.

Furthermore, in the sub-GHz bands, there may be duty cycling restrictions that govern node transmissions [69]. These regulations can limit nodes to as little as 0.1% duty-cycling, depending on the frequency band and transmission power, and can cause delay in retransmissions, increasing end-to-end latency.

2.2.1.4 Summary of Challenges

The following challenges are identified within this overview of low-power wireless mesh networks.

Table 2.2: Summary of challenges identified in Section 2.2.1

Subsection	Challenge
Network Size and Density	• Large hop distances can exacerbate routing complexity and lower end-to-end reliability. • Dense node clusters can cause self-interference in the network.
Topology and Traffic	• Distributed routing schemes are lightweight and resilient, but may under-utilise network resources and create funnelling affects near the sink node. • WSNs have traditionally been designed around data collection applications. IIoT scenarios increasingly require support for multiple services (such as rolling out network updates).
Power and RDC	• Excessive power consumption on nodes can cause orphaned branches. This is a critical issue in Direction-Orientated Directed Acyclic Graph (DODAG) protocols such as RPL, where nodes near the sink serve messages from their children. • Duty-cycling can delay retransmission, particularly in sub-GHz physical layers governed by regulations.

2.2.2 The IEEE 802.15.4 'IoT Stack'

The IEEE 802.15 Working Group (WG) focuses on Wireless Personal Area Network (WPAN) standards such as IEEE 802.15.4-2015 [1] and IEEE 802.15.1-2005 (Bluetooth) [12]. The IEEE 802.15.4 standard, in particular, has been used extensively for communications in industrial-grade monitoring and actuator networks, and is designed to enable devices to last for years at a time, rather than days or weeks. As such, it has emerged as the chosen data link protocol for many Industrial IoT (IIoT) scenarios, and the original IEEE 802.15.4-2003 [70] standard has served as the foundation for a number of well known industrial wireless protocol stacks, such as Zigbee [71], WirelessHART [72], ISA 100.11a [73], and more recently Wi-SUN [74] and efforts from the IETF 6TiSCH [2] WG.

The success of IEEE 802.15.4, together with a number of supporting protocol layers, has cemented its reputation as the IoT communications protocol of choice for many IIoT scenarios, such as industrial wireless control, or SUN and AMI applications. This 'IoT Stack', as shown in Figure 2.8, mirrors the internet networking model used for web applications, and has helped extend IoT to even the most constrained devices. The rest of the section outlines some of the key protocols, standards, and approaches that make up this stack, highlighting how the design and

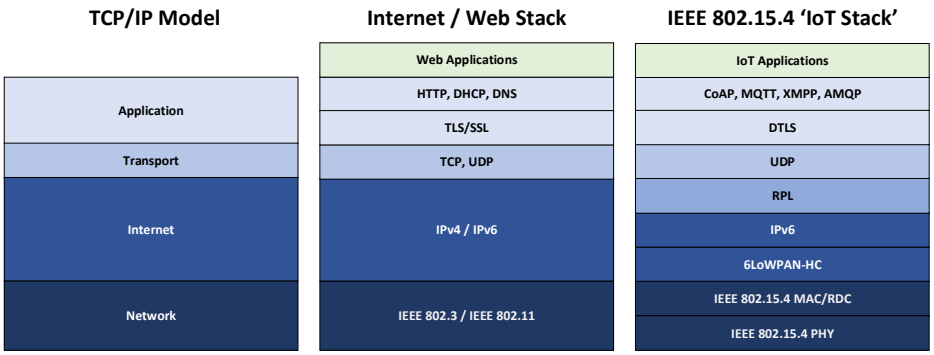


Figure 2.8: The IEEE 802.15.4 'IoT' stack (left) in comparison to the TCP/IP model followed by the internet/web stack (right).

limitations of these layers shape the technical content in later chapters of this thesis. Unless otherwise stated, the work presented in this thesis is based exclusively on 2.4GHz IEEE 802.15.4, although the reader should note that protocols that make up the 'IoT Stack' are equally applicable to the sub-GHz physical layers, which can provide Line of Sight (LOS) ranges of ~100-1000m at the 868 MHz band [75].

2.2.2.1 IEEE 802.15.4 Revisions

A number of major revisions have been released since the base standard proposed in 2003, as well as recent key amendments. The main amendments covered in this thesis are summarised below:

- **IEEE 802.15.4-2003** [70]: Defined two Direct Sequence Spread Spectrum (DSSS) based physical (PHY) layers. A sub-GHz layer for the 868/915 MHz bands (Europe/US), providing data rates of 20 and 40 kbit/s, and a PHY layer operating on the unlicensed 2.4 GHz frequency bands, providing data rates of up to 250 kbit/s. Additionally, it provided beacon-enabled *slotted* CSMA/CA, or non beacon-enabled *unslotted* CSMA/CA at the MAC layer.
- **IEEE 802.15.4-2006** [76]: Improved the data rates of the sub-GHz bands to also support 100 and 250 kbit/s data rates, as well as adding two additional PHY options. In the 868/915 MHz bands, a DSSS approach using either Binary Phase Shift Keying (BPSK) or Offset Quadrature Phase Shift Keying (OQPSK) could be used, or an optional Parallel Sequence Spread Spectrum (PSSS) layer based on a combination of BPSK and Amplitude Shift Keying (ASK). While in the 2.4 GHz band DSSS with OQPSK could be used.
- **IEEE 802.15.4-2015** [1]: Added support for channel hopping and scheduling through the IEEE 802.15.4e TSCH Medium Access Control (MAC) amendment. This was intended to help support deterministic communications for industrial applications, with the frequency

diversity providing resilience against external interference (and multi-path), as well as enabling neighbouring nodes to concurrently transmit on different channels.

2.2.2.2 IEEE 802.15.4 PHY

As outlined in Table 2.3 the IEEE 802.15.4-2015 standard supports an extensive array of different physical layer options, including a number of PHY options to support a variety of application specific areas including: Smart Utility Networks (SUN), Television White Space (TVWS), Low-Energy Critical Infrastructure monitoring (LECIM), and Rail Communications and Control (RCC). As the focus of this thesis is on the application of the SDN paradigm within low-power wireless mesh networks, detailed examination of all these PHY options falls outwith the thesis scope. Nonetheless, there are aspects of the physical layers, in particular the commonly deployed OQPSK-DSSS PHY configuration, that lend themselves to some of the technical work on Concurrent Transmissions (CT) in later chapters.

Table 2.3: IEEE 802.15.4-2015 physical layer summary [1]

PHY	Frequency Bands (MHz)	Modulation	Data rates (kbit/s)
OQPSK PHY	780 / 868 / 915 / 2380 / 2450	OQPSK-DSSS	100 / 250
BPSK PHY	868 / 915	BPSK-DSSS	20 / 40
ASK PHY	868 / 915	ASK-PSSS	250
CSS PHY	2450	DQPSK + 8-ary / 64-ary bi-orth.	1000 / 250
MPSK PHY	780	16-ary orth.	250
GFSK PHY	920	GFSK	100
MSK PHY	433 / 2450	MSK	31.25 / 100 / 250
HRP UWB PHY	sub / 3-10 / 6-10 GHz	BPM + BPSK	110 / 850
LRP UWB PHY	6-9 GHz	OOK + PPM	31.25 / 250 / 1000
SUN OFDM PHY	sub-Ghz / 2450	BPSK / OQPSK / 16-QAM	50-800
SUN OQPSK PHY	868	OQPSK-DSSS	100 / 250

The 127B IEEE 802.15.4 Maximum Transmission Unit (MTU) of the MAC Protocol Data Unit (MPDU), shown in Figure 2.9, has direct implications for control signalling overhead in SDN based low-power wireless networks. Although this was introduced to ensure low packet and bit error rates in lossy environments, it is highly restrictive when considering a full IPv6 stack with multiple layers of headers, and is frequently referenced as one of the key challenges in much of the literature. Similarly, the data rate limitations (250 kbit/s at 2.4GHz, although the recent IEEE 802.15.4q-2016 amendment seeks to introduce higher data rate communications) increases the time taken for each transmission, which then accumulates at each hop to impact end-to-end latencies. Again, this has implications as regards responsiveness and rate of control signalling.

Additionally, a number of recent works [64, 77–80] have explored how the combination OQPSK modulation, alongside the use of spread spectrum techniques, can explain the success of Concurrent Transmission (CT) based flooding protocols when used over IEEE 802.15.4, while

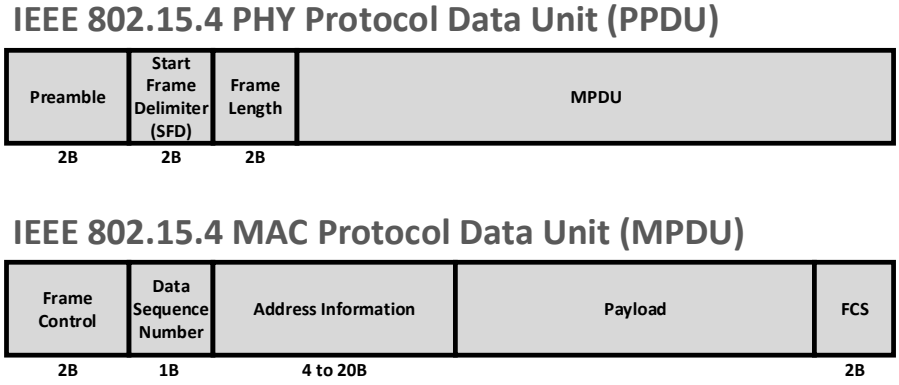


Figure 2.9: IEEE 802.15.4 PHY and MAC frames for the commonly employed 2.4 GHz OQPSK PHY mode.

observing less-forgiving synchronisation and capture requirements in other standards such as the uncoded BLE physical layer. An overview of the principles described in these articles will help the reader to better understand the work presented in Chapter 5.

OQPSK: The IEEE 802.15.4 OQPSK physical layer option prevents the 180° phase changes seen in Quadrature Phase Shift Keying (QPSK), and allows devices to demodulate the signal with simple, non-coherent, frequency demodulators (rather than coherent phase demodulators in QPSK). While this choice was made to reduce cost, the author of [64] the resulting frequency demodulation suffers from beating effects during CT-based flooding due to Carrier Frequency Offsets (CFOs) inherent in device hardware.

DSSS: In Direct Sequence Spread Spectrum (DSSS), the signal is spread over a wider frequency band than required, meaning that narrowband interference has less of an effect on the now greater bandwidth. In IEEE 802.15.4 at the 2.4GHz band, this means that each 4-bit symbol is mapped to a 32 chip symbol (i.e 8 chips to each bit). This physical layer redundancy has particular importance for CTs, where it is suspected that the DSSS combats the previously mentioned beating effects.

2.2.2.3 IEEE 802.15.4 MAC

IEEE 802.15.4-2015 provides two primary MAC layer options, CSMA/CA and TSCH. Until recently, most IEEE 802.15.4 networks were deployed using the base CSMA/CA MAC layer provisioned in the original IEEE 802.15.4-2003 standard. Indeed, in applications that don't consider reliability, or require only a handful of nodes, this is still often the case. However, recent efforts to extend the use of IEEE 802.15.4 to far larger mesh networks require use of the TSCH based MAC in order to maximise channel resources, as well as providing the deterministic communication guarantees needed in many industrial applications.

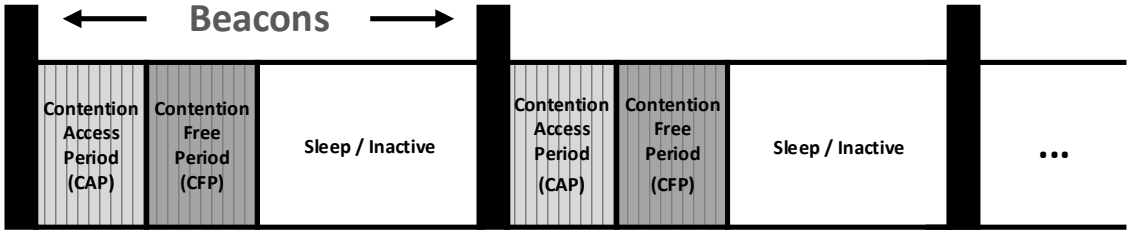


Figure 2.10: IEEE 802.15.4 beacon-enabled *slotted* CMSA superframe.

CSMA/CA: Carrier Sense Multiple Access / Collision Avoidance is an asynchronous contention based MAC layer used in many wireless communications stacks, notably in IEEE 802.11 WiFi. Nodes transmit opportunistically, barring duty-cycle limits, but will first check to see if the channel is clear. If this is not the case, then the node will wait a random period of time (the *backoff* period) before trying to transmit again. In IEEE 802.15.4 the CSMA/CA MAC acts in a slightly different way to the traditional approach taken in IEEE 802.11, doing away with the need for Request to Send (RTS) / Clear to Send (CTS) frames in favour of two different modes: a beacon-enabled *slotted* mode as in Figure 2.10, where a coordinator defines a slotted superframe (synchronised through periodic beacons), or non beacon-enabled *unslotted*, where nodes are free to transmit at any time provided they perform a Clear Channel Assessment (CCA) check and back-off if they find the channel isn't free [1, 81].

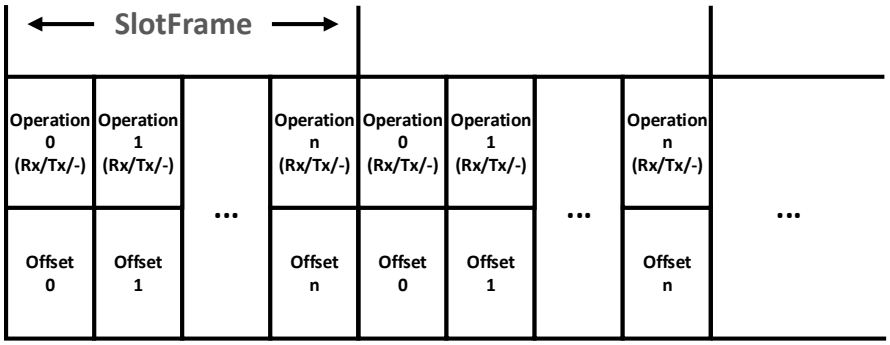


Figure 2.11: A repeating IEEE 802.15.4 TSCH SlotFrame. Each slot is assigned an operation (R_x, T_x, or $sleep$), and a channel offset.

TSCH: Time Scheduled Channel Hopping (TSCH) is a deterministic MAC layer that introduces frequency hopping across a Time Division Multiple Access (TDMA) schedule. This frequency diversity allows for greater reliability and energy efficiency than CSMA-based MAC layers, particularly in environments where there is a high degree of fading and interference.

Figure 2.11 shows how the TSCH schedule is sliced into repeating SlotFrames, consisting of timeslots spread over a number of frequency channels. Nodes periodically exchange synchronisation beacons, which contain startup information about the TSCH schedule. Each timeslot can be allocated as a dedicated transmit (T_x) or receive (R_x) opportunity for that node, used as a *sleep* cell, or *shared* with other nodes in a contention-based scheme. By creating a schedule across the spectrum, as demonstrated in Figure 2.12, self-interference (i.e. collisions) in the network is reduced, as it ensures that co-located nodes aren't attempting to transmit simultaneously. In addition, nodes only need to be awake for T_x, R_x, and *shared* cells, allowing them to sleep at all other times and improving their energy consumption.

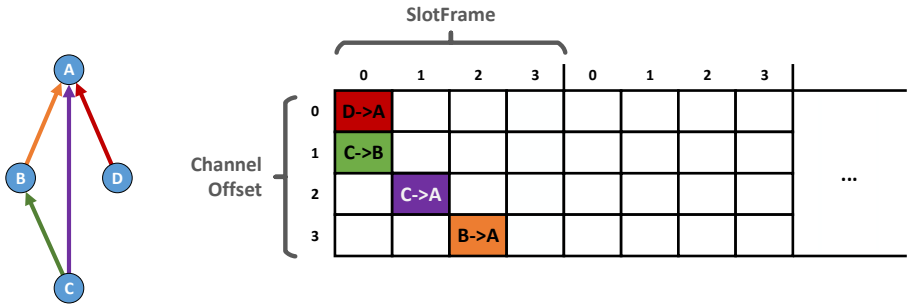


Figure 2.12: A minimal TSCH schedule using channel diversity to schedule co-located transmissions.

Although TSCH can be used in an opportunistic manner, as in the Wi-SUN Field Area Network (FAN) stack [74] which can[1] operate using the IEEE 802.15.4e-2012 [83] TSCH layer on a Carrier-Sense Multiple Access (CSMA) basis (i.e. on a contention based scheme without any scheduling), IETF 6TiSCH [2] defines mechanisms to create optimal schedules and distribute these schedules to network node. These efforts are described in the latter part of this section.

2.2.2.4 6LoWPAN

The Internet Engineering Task Force (IETF) 6LoWPAN WG [84] was set up with the aim of introducing IPv6 to low-power, low-cost IEEE 802.15.4 devices. This was driven by the goal of extending Internet Protocol (IP) connectivity to even the most constrained devices, allowing them to interconnect and communicate with other IP networks. Given the limitations of IEEE 802.14.5 WPAN networks (such as a 127B MTU on the 2.4 GHz PHY layers), three key goals were identified:

- **6LoWPAN-HC** [85]: Introduce mechanisms for IP Header Compression (HC), as well as support for fragmentation and reassembly.

[1] In practicality, current Wi-SUN FAN research mainly reverts to *unslotted* CSMA [82]

- **6LoWPAN-ND** [86]: Establish means to perform IPv6 Neighbour Discovery (ND) and address auto-configuration.
- **6LoRH** [87]: Define Routing Headers (RH) for use in 6LoWPAN *route-over* topologies, allowing RPL Option compression.

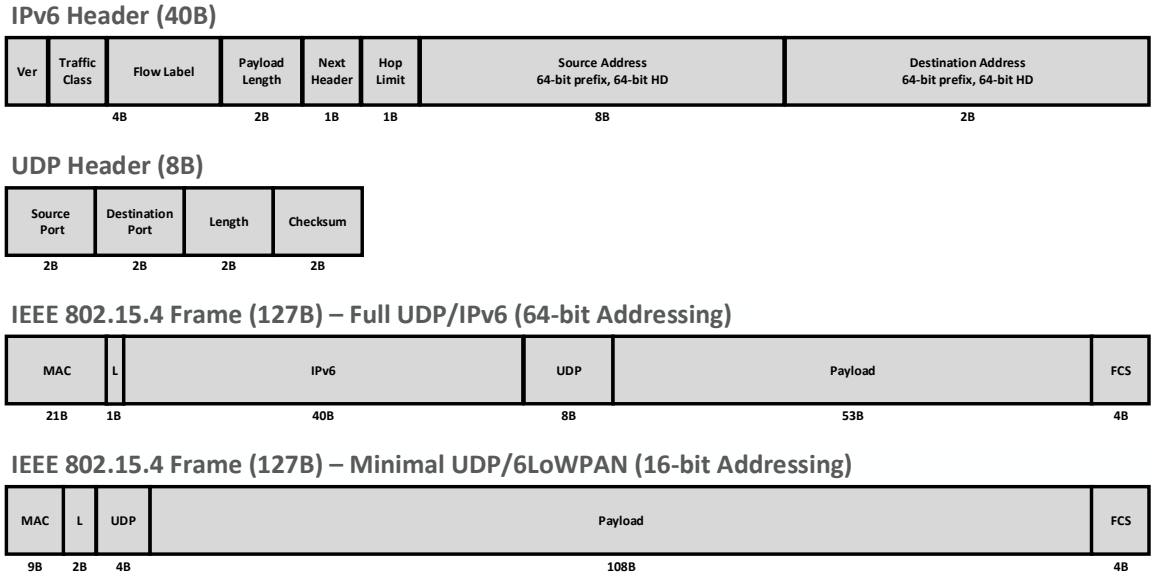


Figure 2.13: IPv6 and UDP headers, and the examples of full IPv6 and minimal 6LoWPAN header compression frames.

Supporting the 'I' in IoT, 6LoWPAN enables the end-to-end transmission of 1280B IPv6 datagrams in a IEEE 802.15.4 network where packets are constrained by a 127B MTU. Figure 2.13 shows how, if the full 48B UDP/IPv6 headers are used, then this only leaves a mere 53B payload for application data. To address this, 6LoWPAN firstly compacts the 40B IPv6 and 8B UDP headers by compressing or removing fields. If the payload size forces the packet over 127B, then 6LoWPAN also supports fragmentation of the larger 1280B IPv6 MTU. Reconstitution of the packet is established on a hop-by-hop basis as outlined in the original RFC [84] or, alternatively, recent work outlines how reassembling might be achieved solely at the destination node [88].

Although the 6LoWPAN standard brought IP connectivity to even the smallest IoT devices, it comes with a caveat. Even though fragmentation works in principle, the hardware constraints placed on nodes within the low-power wireless mesh means that, in practice, they have limited RAM to be able to buffer fragmented packets for later reassembly. Additionally, by its very nature fragmentation imposes a cost on the end-to-end latency of packets, as each fragment must be transmitted across multiple hops: either fragmenting and reassembling it at each hop, or trusting that all fragments eventually arrive at the intended destination and reassembling at delivery.

2.2.2.5 RPL

Unlike wired networks, where the routing topology is imposed by the physical wires, low-power and lossy networks such as IEEE 802.15.4 do not typically have predefined topologies. As a result, there exist protocols allowing nodes to self-organise and establish ad-hoc connections. Over the past decade RPL [24] has provided a lightweight and distributed mechanism allowing mesh networks to both discover and maintain routing topologies, and now forms integral part of many low-power wireless networks. RPL's success stems from its use of a particular form of Directed Acyclic Graph (DAG), a DODAG, in order to create topologies where information is forwarded towards a single location. An example of such a graph is illustrated in Figure 2.14, which shows how DODAG nodes only need to maintain knowledge of their parents in order to reach a single root node, R.

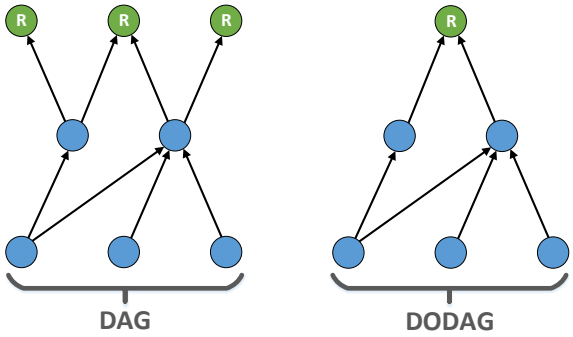


Figure 2.14: A Directed Acyclic Graph (DAG) (right) versus a Direction-Orientated Directed Acyclic Graph (DODAG) with a single root node (left), as formed by RPL.

There are a number of terms associated with RPL that are introduced in this section and used in later chapters. In order to provide a reference for the reader these are summarised below in Table 2.4, while the remainder of this section on RPL outlines how the protocol both constructs a distributed routing topology.

Table 2.4: Summary of RPL terms.

Term	Description
Direction-Orientated Directed Acyclic Graph (DODAG)	A tree-like graph with no cycles, and single root node with no outgoing edge (although this often acts as a border router).
DODAG Information Solicitation (DIS)	ICMPv6 message used by nodes to request RPL DAG information from one-hop neighbours.
DODAG Information Object (DIO)	ICMPv6 message sent as a response to DIS messages.
Destination Advertisement Object (DAO)	Sent from child nodes to the parent or root (depending on the RPL mode) in order to advertise themselves as a destination in the DAG.
RPL Rank	Indicates a node's position in the graph relative to the DODAG root.
RPL Storing Mode	Nodes maintain a routing table for their Sub-DODAG.
RPL Non-Storing Mode (RPL-NS)	Nodes only know their parent, and the root keeps a routing table for the whole DODAG.

Topology Construction: The step-by-step process of DODAG construction is shown in Figure 2.15. To establish an *upwards* connection, nodes can solicit DODAG information from neighbouring nodes through a multicast DODAG Information Solicitation (DIS) control message. Neighbours will respond to this request with a DODAG Information Object (DIO) message containing information about the DODAG and RPL instance. The initiating node can then choose to join the DODAG by adding the sender to its parent list, and computing its *rank* relative to the parent node based on an objective function such as Minimum Rank Objective Function (MRHOF) [89] or Objective Function Zero (OF0) [90]. Once the parent has been chosen, an *upwards* link has then been established to the root node, R.

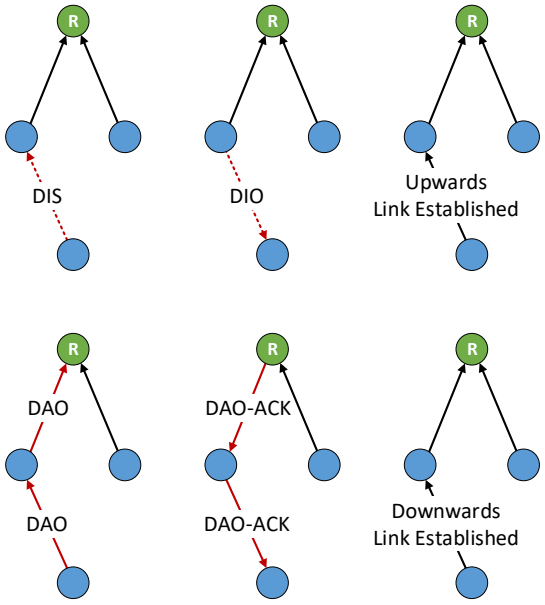


Figure 2.15: Example RPL DODAG construction. DIS and DIO messages form upwards links. DAO messages inform R that the new node is reachable in order to establish a downward link.

To establish the reverse *downwards* link from the DODAG root, the newly joined node informs parents that it is reachable by sending a Destination Advertisement Object (DAO) control message upwards until it reaches a node capable of maintaining routing state. In RPL Non Storing Mode (RPL-NS), this is always the DODAG root node, however RPL Storing Mode allows routing through the common parent of child nodes (though this requires nodes with enough memory to store and maintain these tables). Nodes incapable of maintaining routing tables (i.e. all nodes *except* the root in RPL-NS), attach a next hop address to the DAO control message and forward it on to the next parent. When it reaches a routing node, the path attached to the DAO header is stored in the routing table, and a DAO-ACK message is sent back to the newly joined node as confirmation that the *downwards* link has been established. Figure 2.16 shows a completed 4-hop DODAG where nodes are organised in accordance with their rank.

Yet the question of how to form the RPL DODAG topology, i.e. which parent to choose and

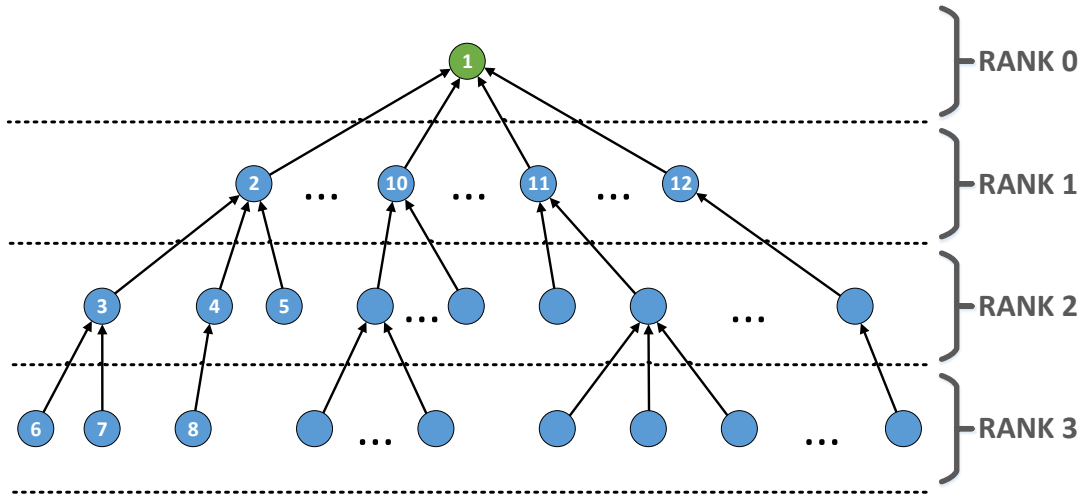


Figure 2.16: Hypothetical example of completed DODAG graph and tree topology formed by RPL, using hop count as a link metric. A node's rank gives that node's position within the graph with respect to the DODAG root.

what metrics to consider in making that decision, continues to be an outstanding research issue; particularly in the face of TSCH based MAC protocols such as those found in the 6TiSCH networking stack. Specifically the tree-like graph created by the DODAG results in a higher utilisation of resources the closer a node is to the root, as it serves messages routed from its children. In large-scale IIoT networks served by these two standards, where nodes can number in the thousands, this can cause considerable congestion and contention at the DAG root.

2.2.2.6 Summary of Challenges

The following challenges are identified within this overview of the IEEE 802.15.4 'IoT Stack'.

Table 2.5: Summary of challenges identified in Section 2.2.2

Subsection	Challenge
IEEE 802.15.4 PHY	• Low data rates (up to 256Kb/s at 2.4GHz) and a small MTU (127B) restrict capacity for additional communications overhead. Sub-GHz offers increased range, but at even lower data rates (40 kbit/s BPSK). • Increased range (kilometres at sub-GHz) and a small MTU (127B) restrict capacity for additional communications overhead.
IEEE 802.15.4 MAC	• Asynchronous CSMA transmissions can result in contention in dense networks. • Reception misses can result in high latencies due to RDC + channel hopping in TSCH.
6LoWPAN	• IPv6 fragmentation can result in longer hop-to-hop delays and a high end-to-end latency. • Hardware limitations can result in insufficient buffer space for effective fragmentation.
RPL	• Funnelling effects in the RPL topology forces nodes near the root to serve a higher proportion of messages than child nodes, increasing contention and depleting energy. • Routing tables are held at either the sink or common parents, resulting in sub optimal routing solutions for anything other than direct communications between a node and the sink (i.e. point-to-point communications can be difficult).

2.2.3 6TiSCH Industrial IoT

As covered in Section 2.2.2, a Time Scheduled Channel Hopping (TSCH) MAC layer was incorporated into the IEEE 802.15.4-2015 standard update [1]. It provided an alternative to the asynchronous CSMA/CA channel access modes governing previous versions, allowing IEEE 802.15.4 to benefit from frequency diversity as well as temporally scheduled communications. However, IEEE 802.15.4e-2012 didn't specify a mechanism for building and maintaining the TSCH schedule, leaving it to the higher layers to define either a static schedule, or allow nodes to contest slots on a CSMA basis.

This 'standardisation gap' prompted the creation of the IETF 6TiSCH WG [2], which aims to bridge this gap by defining how the schedule is created. Not only does it borrow aspects from the WirelessHART [91] and ISA100.11A [73] standards for industrial wireless control,

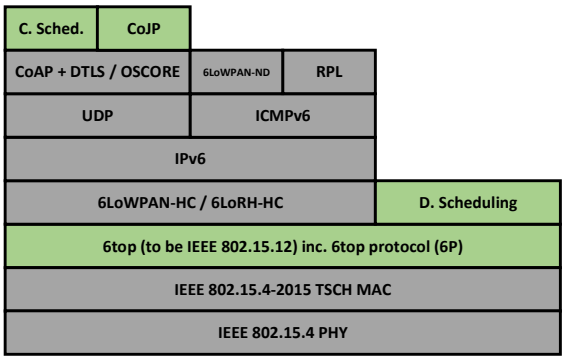


Figure 2.17: 6TiSCH Industrial IoT (IIoT) stack [2]. 6TiSCH layers are highlighted in green.

but it also incorporates concepts from the IETF Deterministic Networking WG [92] as well as drawing on activities from the IETF SDN WG [5]. Figure 2.17 details the full 6TiSCH stack for IEEE 802.15.4 low-power wireless networks (as of June 2019, version 20 of the draft IETF 6TiSCH architecture [2]). This section attempts to introduce the main concepts of 6TiSCH and its associated terminology, as well as explore how the SDN concept is imagined within the architecture.

2.2.3.1 General Approach

The 6TiSCH standard is primarily focused with producing specifications for how nodes construct and communicate the TSCH schedule, establishing default scheduling functions for dynamic scheduling of timeslots for IPv6 traffic, and defining an interface to enable deterministic routes (as defined by IETF [92]) across the 6TiSCH network through the manipulation of 6TiSCH scheduling and forwarding mechanisms. In this fashion, 6TiSCH is able to offer service guarantees and non-competing communications for Industrial IoT applications through efficient allocation of node and radio resources across the mesh, allowing the network to scale through well-scheduled time and frequency diversity.

To achieve these goals, a scheduler orchestrates communications over individual links in an optimised and non-conflicting manner by allocating slots within the TSCH MAC layer (as illustrated in Figure 2.18). In the time domain the schedule operates in Time Division Multiple Access (TDMA) manner, whereas in the frequency domain it divides the wireless spectrum into multiple channels. The scheduling period is referred to as a *slot-frame*, which repeats over time. A schedule is formed by assigning *timeslot* and channel offsets to each communication link, and specifying which node should transmit or receive data to/from its scheduled counterpart within a *slot-frame*. Channel-hopping is also adopted, where communication links hop over a set of available channels according to pseudo-random channel hopping schemes, therefore mitigating the effect of narrow-band interference and multi-path fading.

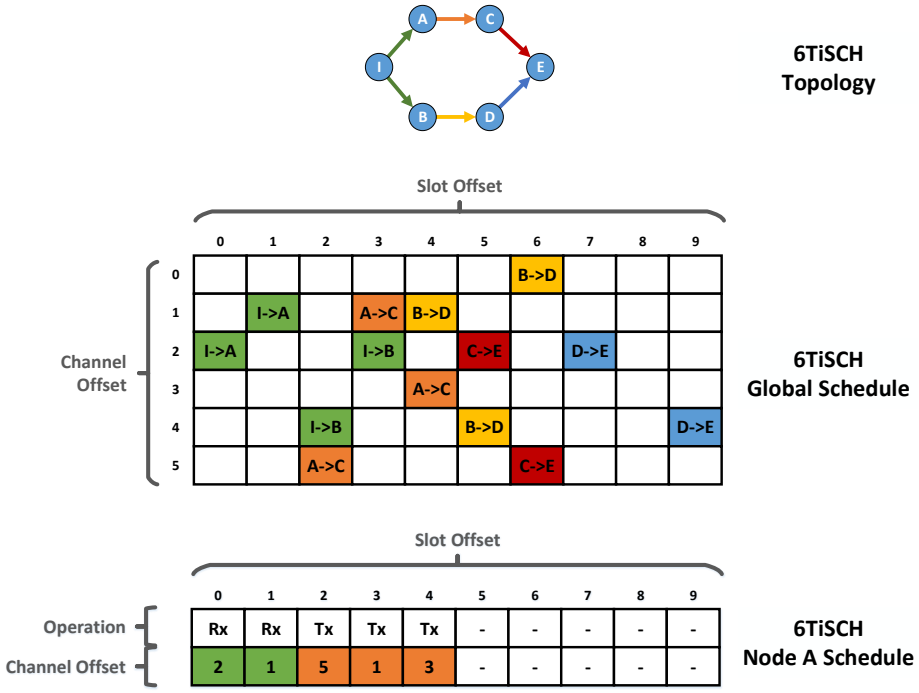


Figure 2.18: Example 6TiSCH topology and schedules. Either a centralised or distributed scheduler creates a global schedule, and assigns an operation + channel offset to each local node schedule.

2.2.3.2 Schedule Management

The 6TiSCH standard allows for a number of schedule management paradigms. These mechanisms allow for flexible maintenance of the TSCH schedule, based upon the needs of the network.

- In *Static Scheduling* a common fixed schedule is shared by all nodes in the network, and can be used as a network initialisation mechanism. It is equivalent to Slotted ALOHA [93], with the exception that TSCH 10ms slot times can be larger than the frame transmission time (up to ~4ms at 256kbps OQPSK), and remains unchanged once a node joins the network.

- *Neighbour-to-Neighbour* allows distributed scheduling functions by providing mechanisms so that matching portions of the TSCH schedule can be established by and between two neighbouring nodes. As part of the charter, 6TiSCH is tasked with incorporating elements of SDN architecture: for *Remote Schedule Management*, a centralised Path Coordination Engine (PCE) is able to make schedule changes based on the overall network state collected from each node.

- A *Hop-by-Hop* scheduling mechanism is used for the distributed allocation of 6TiSCH tracks. With Hop-by-Hop scheduling, a node can reserve a slice of the TSCH schedule as

a dedicated Layer-2 forwarding path towards a destination, by getting 6top to reactively allocate cells at each intermediate node on a Peer-to-Peer (P2P) basis.

2.2.3.3 Control Signalling

The 6TiSCH Operation Sublayer (6top) allows 6TiSCH to communicate TSCH scheduling information to and from nodes within the network, and provides mechanisms for both distributed and centralised scheduling. In the distributed scenario, schedule information is communicated between nodes using the 6top Protocol (6P), transported on TSCH Information Elements (IEs). In a centralised scheduling scenario, schedule information is communicated to and from a centrally located Path Computation Engine (PCE), as defined by the IETF DetNet WG [92] and somewhat analogous to an SDN controller, through the 6top CoAP Management Interface (COMI).

2.2.3.4 Routing and Forwarding

Figure 2.19 shows the three forwarding models supported by 6TiSCH: *IPv6 Forwarding*, where each node decides on the forwarding path based on its own forwarding tables; *6LoWPAN Fragment Forwarding*, where successive fragmented packets are forwarded onto the destination of the first fragment; and finally, *Generalised Multi-Protocol Label Switching (G-MPLS) Track Forwarding*, which switches frames at Layer-2 based on dedicated ingress and egress cell bundles along a path. IPv6 and 6LoWPAN fragment forwarding, which deal with non-deterministic traffic, are routed using the RPL [24] distributed routing protocol, while it was expected that G-MPLS track forwarding would be facilitated through a mixture of centralised routing at the PCE, and distributed hop-by-hop routing.

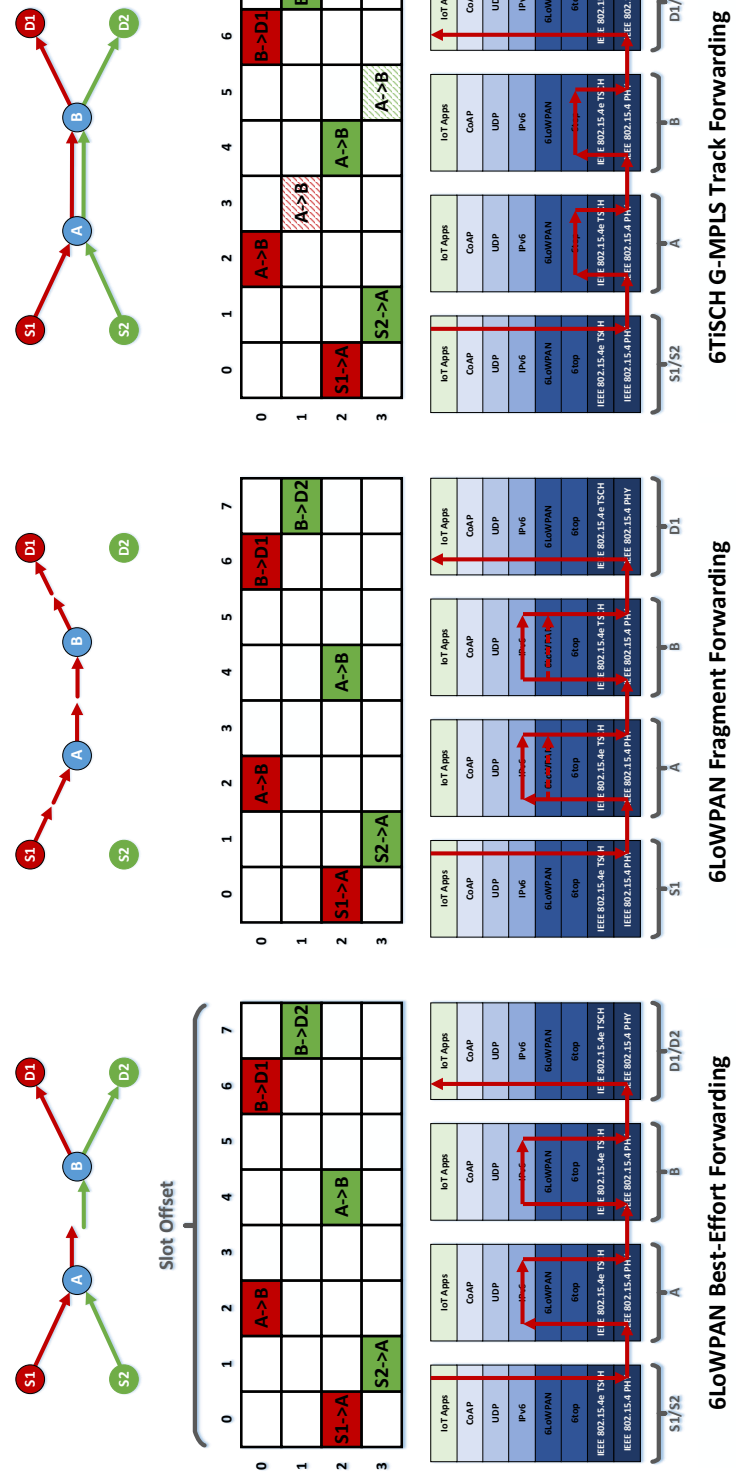


Figure 2.19: 6TiSCH forwarding from left to right: 6LoWPAN 'best-effort' forwarding, 6LoWPAN fragment forwarding, and 6TiSCH G-MPLS *track* forwarding, where the flow S1→D1 reserves dedicated slot resources (shaded) to complete a deterministic end-to-end path.

2.2.3.5 Summary of Challenges

The following challenges are identified within this overview of the IETF 6TiSCH standard.

Table 2.6: Summary of challenges identified in Section 2.2.3

Subsection	Challenge
Schedule Management	• Centralised v.s. distributed. • Allocation of resources to flows.
Routing & Forwarding	• Mesh-under vs route-over. • Fragmentation in a TSCH environment.

2.2.4 Conclusions

This section has provided a brief overview of three key areas that impact the application and design of SDN architecture in a low-power wireless mesh environment: firstly, *what is a mesh network and what are the associated challenges?*; secondly *how does the IEEE 802.15.4 standard and low-power 'IoT Stack' constrain the network*; and finally, *why is IETF 6TiSCH Industrial IoT (IIoT) important to SDN in low-power wireless?* From this overview, a number of challenges have been identified and, in light of these, conclusions are drawn with reference to Research Questions [Q2, Q4]:

[Q2]: *How does SDN control signalling affect the low-power wireless mesh?* The centralised v.s. distributed scheduling question in IETF 6TiSCH, in addition to evaluation of 6LoWPAN fragmentation forwarding, provides a model for how the increased communications can affect a low-power wireless mesh network. Specifically, the size, density, traffic characteristics, and topology of the network all have a bearing on how additional communications will affect network performance. Although these aspects can be configured for a particular application or traffic pattern, the multiple and varied requirements of SDN make this particularly challenging.

[Q4]: *Can SDN scale in a low-power wireless mesh?* The TSCH MAC adopted by IETF 6TiSCH enables scheduling of concurrent orthogonal communications within the mesh. By reducing collisions this allows mesh networks to scale and is of particular importance to the large-scale IIoT networks targeted by the standard.

2.3 Concurrent Transmissions and Synchronous Flooding

Recent research efforts have explored using Concurrent Transmissions (CT) to enable high-reliability, low-latency Synchronous Flooding (SF) protocols for low-power wireless mesh networks [94]. This section introduces necessary background on CT and SF, which form the basis for the technical content presented in Chapter 5.

Firstly this section summarises a number of physical layer phenomena which allow reliable demodulation of packets sent by concurrently transmitting, co-located nodes. These phenomena are shown in Figure 2.20. Although CT in low power wireless has been established as a highly capable solution for applications that require extremely high reliability and low latency (such as wireless control systems) [3, 7, 8, DC1, DC2], the physical layer properties behind this success have often been poorly understood by the community, and are only now being researched. Secondly this section covers an overview of how CT can be used to create SF protocols. These can rapidly disseminate data across the network using a combination of aggressive temporal, frequency, and spacial diversity, and can be considered in two parts.

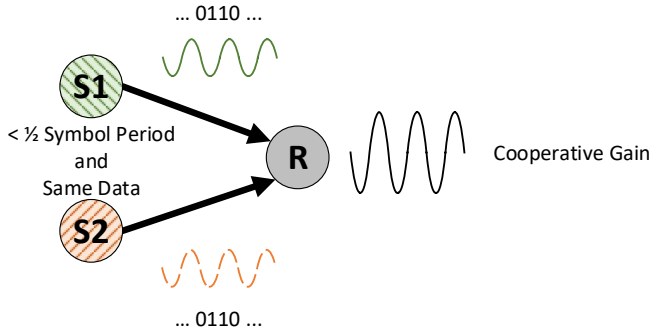
1. Lower layer SF primitives that establish a schedule of communications in each flood.
2. Higher layer SF protocols that govern the behaviour of nodes both during, and between floods.

2.3.1 Concurrent Transmissions

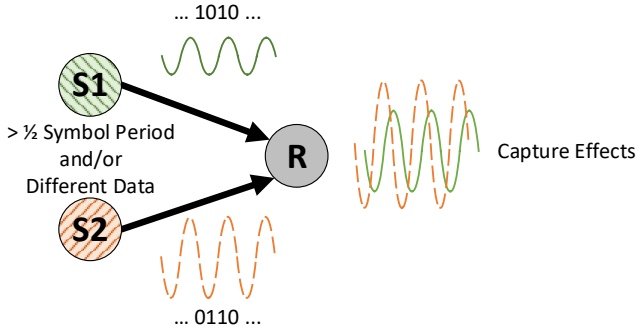
Concurrent Transmissions (CTs) describes the process in a wireless network whereby two or more nodes within range of each other simultaneously attempt to transmit data on the same frequency. Although one would typically consider these signals to be destructive due to the effects of multipath, the authors of Glossy [6] showed that by utilising CT to transmit the same data packet in a single flood, they could improve network reliability and synchronisation, and used CT to achieve highly reliable *one-to-many* communication within multi-hop low-power mesh networks. They found that as long as the maximum temporal displacement between concurrently transmitted signals (of the *same* data) was less than half a symbol period (($\Delta_{max} > 0.5\mu s$) in IEEE 802.15.4), then that data can be reliably demodulated without the transmitted signals interfering with one another. A sizeable research community has been built around the work presented in Glossy. Not only did the authors successfully demonstrate that CT could be used to achieve highly reliable *one-to-many* communication within IEEE 802.15.4 low-power mesh networks, the Glossy code was subsequently made publicly available by the authors, and has served as the basis for much of the CT research since.

2.3.1.1 'Constructive Interference' and Beating Effects

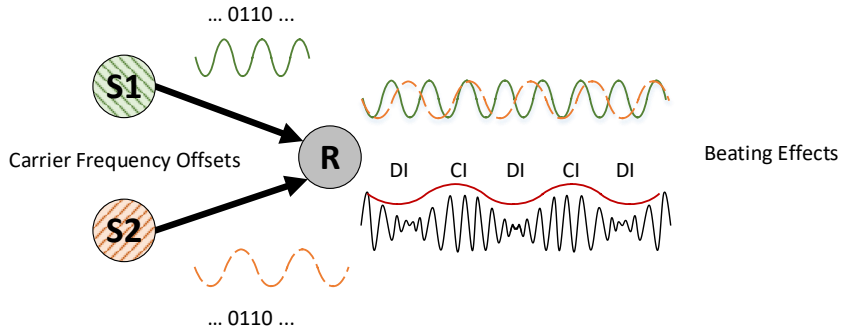
The original Glossy paper demonstrated that when multiple nodes simultaneously transmit the *same* data, a gain was seen at the receiver (Figure 2.20a). To date, most of the literature around CTs has attributed this gain to so-called 'constructive interference'. However, recent research has shown that this assumption was not entirely correct. Specifically, when there are few transmitters, packets will experience a mix of constructive and destructive interference due to beating effects as shown in Figure 2.20c. This is the result of Carrier Frequency Offsets (CFOs) between the oscillators of transmitting nodes, causing a beating envelope of energy maxima



(a) Two concurrent transmitters synchronised to within half a symbol period (0.5 μs in IEEE 802.15.4) will provide Cooperative Gain at a receiver when sending the same data.



(b) Capture Effects can affect concurrent transmitters where one signal is received with significantly more power than the others (3dBm in IEEE 802.15.4).



(c) Beating Effects are prominent in CTs with only a few transmitters. The packet envelope experiences both Constructive (CI) and Destructive (DI) interference due to Carrier Frequency Offsets (CFOs) between the transmitters.

Figure 2.20: A number of physical layer phenomena occur in Concurrent Transmissions: (a) Cooperative Gain (termed as so-called 'Constructive Interference' in literature), (b) Capture Effects, and (c) Beating Effects.

and minima across the packet. This thesis proposes 'Cooperative Gain' as a more appropriate term than 'constructive interference', as the received gains depend directly on transmitters being synchronised in *both* time *as well as* frequency, and can be experienced differently across the packet.

2.3.1.2 Capture Effects

When multiple initiators transmit *different* data, then CT relies upon the Capture Effect [65] to reliably receive data at the destination. This refers to the phenomenon that the strongest signal out of multiple co-channel signals will be demodulated (Figure 2.20b). It occurs either if one of the received signals is around 3dB stronger (although this depends on the particular hardware and modulation schemes used) or if one of the signals is received significantly earlier than the other competing signals. In each case, this allows the receiving radio to lock on to the preamble of the stronger signal. Although the other signals may still interfere to cause errors, there is a high probability that one of the transmissions will be demodulated.

2.3.1.3 CT Feasibility in Other Physical Layers

Concurrent Transmissions have been used to great effect as the basis for *one-to-all*, *one-to-many*, and *many-to-one* SF data collection protocols in IEEE 802.15.4. However, recent research has shown its applicability as a flooding solution not only for Bluetooth [78, 95], but also in LoRa, and as the basis for localisation techniques in Ultra Wide Band (UWB) [96]. Yet there is doubt as to how well CT-based *many-to-one* protocols would perform on physical layers other than OQPSK-DSSS. Indeed, the success of CTs in present literature has been attributed to a mixture of non-coherent frequency demodulation in low-power receivers, as well as help from Direct Sequence Spread Spectrum (DSSS), where capture effects in IEEE 802.15.4 help receivers demodulate a packet when one signal dominates [64, 77, 78]. In particular, *many-to-one* approaches such as Crystal [3] rely on the concept of a shared flood, where multiple initiators attempt to send different data. If multiple source nodes send data within the same flood, this precludes nodes from benefiting from Cooperative Gain, meaning the protocol is heavily reliant capture effects at receivers. Notably, the authors of [78] experimentally demonstrated that when nodes transmit different data and CTs are applied to Bluetooth physical layers (which do not experience capture effects as significantly as IEEE 802.15.4), then reliability drops significantly.

2.3.2 Low-Layer Synchronous Flooding Primitives

Synchronous Flooding (SF) primitives define the MAC and data link properties used within a single flood: such as radio timings, offsets, guards, and the schedule of transmission (T_x) and receive (R_x) slots. This subsection describes the general operation of SF primitives, and outlines two approaches used to maintain accurate synchronisation between nodes.

2.3.2.1 General Operation

With reference to the back-to-back T_x approach in Figures 2.23 and 2.24, the length of each timeslot (T_{slot}) is determined by the time needed to transmit the packet (T_{tx}) (i.e. made up of the preamble, Start Frame Delimiter (SFD), MAC Protocol Data Unit (MPDU), and packet data), a software delay (T_{sw}) introduced by the micro-controller, a radio calibration delay (T_{cal}), and a processing delay incurred by the receiver radio (T_{rs}) incurred by the hardware, s.t.

$$T_{slot} = T_{tx} + T_{sw} + T_{cal} + T_{rs} \tag{2.1}$$

Non-initiating nodes listen for flood transmissions. When they successfully receive a packet, a relay counter in the header indicates how many hops (and consequently how many slots) have elapsed. Nodes combine this with their knowledge of T_{slot} to calculate the reference time of the initiator and synchronise to that node. After synchronisation, they relay the packet on the next timeslot; alongside any other neighbours who also received within that slot. Nodes at the next hop will repeat the process, and so on, until nodes have either reached MAX_TX or the flooding period (Δ_{SF}) has elapsed (calculated from T_{slot} and MAX_SLOTS). At each R_x slot, nodes resynchronise to the flood in order to compensate for drift and determine the timing of the next T_x slot. Once all nodes have been synchronised in the initial flood, the initiating node effectively acts as a timesync for the network, allowing non-initiating nodes to duty-cycle their radio. Guards are set on receiving nodes so that they wake up before the next flood with enough time to set their radios to R_x.

Each flooding period is partitioned into slots. The maximum number of slots (MAX_SLOTS) and maximum number of transmissions (MAX_TX) are statically configured at the start of each flooding round. At the start of each flooding round the initiating (source) node transmits a packet, and repeatedly transmits on every slot until MAX_TX. All other nodes have set their radios to receive. The receiving node then relays the packet on the next slot, concurrently transmitting with all other forwarding nodes.

MAX_SLOTS is used to calculate the maximum flooding time, while MAX_TX is the number of times a node concurrently transmits after the first reception. Factors such as external interference, poor connectivity, and the network hop distance need to be taken in to consideration when these variables are set. Increasing them allows for greater reliability, and at a minimum the number of slots needs to equal the hop distance of the network, while minimising them allows for lower end-to-end latency in protocols with multiple flooding periods, as well as reducing energy consumption. In essence, these values are a trade-off between latency and allowing greater temporal and frequency diversity (if paired with slot-by-slot channel hopping [7]).

2.3.2.2 Interleaved R_x/T_x Synchronisation

The radio-driven nature results in clock drift, meaning synchronisation can not be reliably maintained over multiple transmissions. As demonstrated in Figures 2.21 and 2.22, the authors

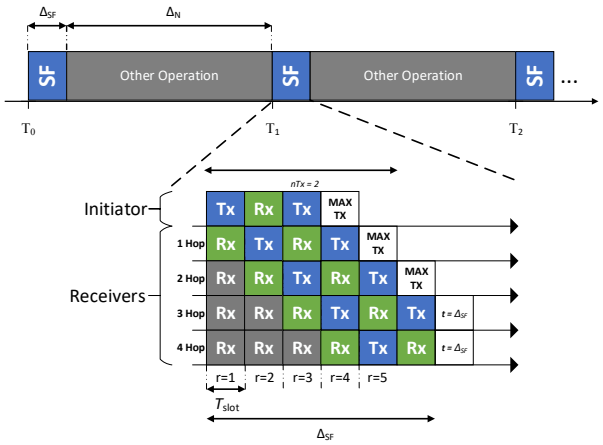
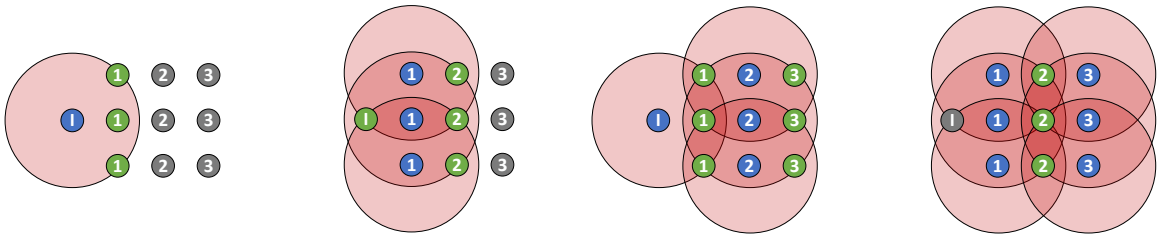


Figure 2.21: Interleaved R_xT_x approach as used in the original Glossy [6] paper.



(a) $r = 1$ (b) $r = 2$ (c) $r = 3$ (d) $r = 4$

Figure 2.22: Synchronous flooding using back-to-back transmissions in a 3-hop network (based on the schedule in Figure 2.21. Blue indicates a transmission, whilst green indicates a reception.

of Glossy proposed interleaving transmission (T_x) and reception (R_x) slots so that successful receptions could help correct this drift and align the next slot.

2.3.2.3 Back-to-Back (B2B) T_x Synchronisation

Recent approaches have demonstrated techniques to estimate this drift [7, DC2], and that slot interleaving is not necessary. This allows nodes to repeatedly T_x after the first reception so that data is forwarded at every slot, meaning the time taken to fully propagate the packet across the network is substantially reduced. This back-to-back T_x approach is demonstrated in Figures 2.23 and 2.24.

2.3.3 High Layer Synchronous Flooding Protocols

The initial Glossy research has inspired a slew of novel SF protocols and approaches that have consistently outperformed other low-power wireless solutions in both reliability and latency metrics at the IEEE Embedded Wireless Systems and Networks (EWSN) Dependability Competition. A comprehensive review of SF-based protocols can be found in a recent survey and tutorial of CT

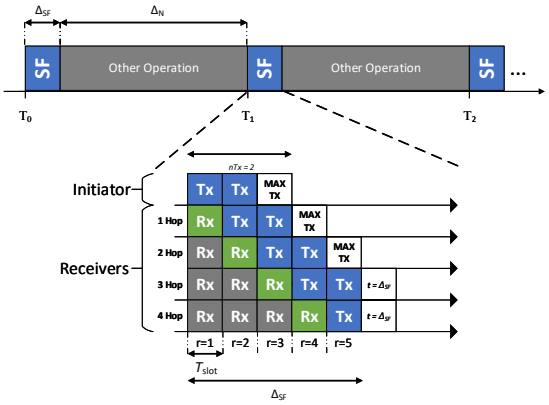
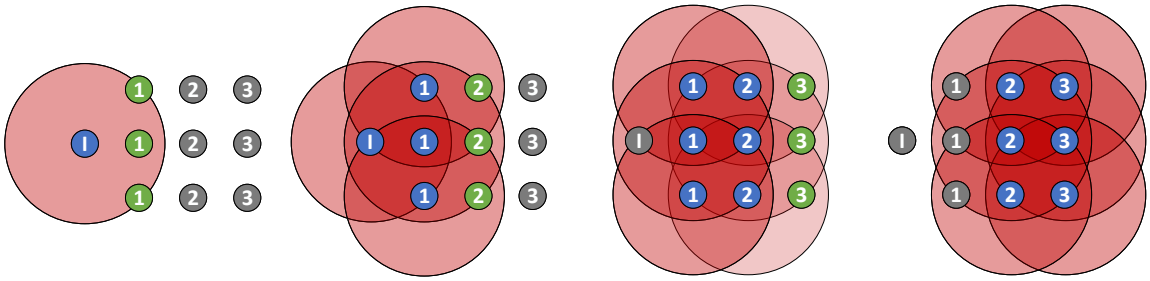


Figure 2.23: Back-to-back T_x approach used in more recent SF protocol solutions [7–9], and in the technical works supporting this thesis [C3, DC1, DC2, J1]. As $nT_x = 2$, transmissions are repeated twice after initial reception.



a) $r = 1$ b) $r = 2$ c) $r = 3$ d) $r = 4$

Figure 2.24: Synchronous flooding using back-to-back transmissions in a 3-hop network (based on the schedule in Figure 2.23. Blue indicates a transmission, whilst green indicates a reception. As $nT_x = 2$, transmissions are repeated twice after initial reception.

in IEEE 802.15.4 networks [94], however the overview presented below summarises some of the most notable approaches to date.

2.3.3.1 *One-to-Many* (Dissemination) Protocols

As the *one-to-many* (or *one-to-all*) traffic pattern is inherent within a R_x/T_x low-level SF primitive [6], or B2B T_x primitive [7], research into SF protocols that facilitate these patterns has tended to focus on how to improve reliability and latency within the SF primitives themselves.

2.3.3.2 *Many-to-One* (Collection) Protocols

Crystal [3] is a highly reliable and energy-efficient data collection protocol based on Glossy flooding. Recent work on Crystal has enhanced the base protocol with improvements to interference management [97].

2.3.3.3 *Many-to-Many* Protocols

The authors of Mixer [98] utilise Random Layer Network Coding (RLNC) to support the full spectrum of *one-to-all* to *all-to-all* communications and approaches order-optimal scaling $O(M+T)$, compared to $O(M.T)$ in sequential flooding solutions. Like Crystal, Mixer uses shared floods with multiple initiators; however, rather than relying on the capture effect to receive at the destination, or perform M sequential messages like in LWB [99], Mixer nodes combine packets using RLNC. This allows multiple destinations to receive all packets from multiple initiators within a far shorter time in comparison to conventional SF solutions.

2.3.3.4 Consensus Protocols

Chaos [100] is a similar proposition to Crystal, in that it makes use of the capture effect to allow multiple nodes to transmit different data within a single (shared) flood. However, Chaos provisions a short period of time in between slots on its R_x/T_x lower layer SF primitive to allow each node to aggregate the recently received data with previously received packets. This allows the network to aggregate data from several nodes, achieving consensus across the whole flood.

The Chaos concept is taken further in A^2 [101], where the authors extend the protocol to build a synchronous transmissions based communication layer supporting multi-phase consensus protocols, as well as a number of core network services such as network association, hopping sequence distribution, and re-keying.

2.3.3.5 Multi-Protocol/Adaptive Approaches

Low-Power Wireless Bus (LWB) [99] takes a deterministic approach to synchronous flooding in that it provides a scheduling mechanism which supports several traffic patterns through the use of a shared protocol bus.

Baloo [102] establishes an abstract middleware layer that allows the creation of (theoretically) any SF protocol, as long as the correct SF primitives are available for the target hardware. Baloo is concurrent work to the Atomic-SDN architecture presented in Chapters 5, where both were presented as part of the IEEE EWSN 2019 conference and competed in the co-located Dependability Competition.

2.3.4 Conclusions

From this overview of Concurrent Transmissions (CT) and Synchronous Flooding (SF), conclusions are drawn with reference to Research Questions [Q2, Q3, Q4]:

[Q2]: *How does SDN control signalling affect the low-power wireless mesh?* The concurrent nature of SF means that all nodes must participate in the flood, whether as initiators, receivers, or merely forwarders. However, this flood is bounded by nature, meaning that the network has consensus on when this flooding period will end. Although network resources are blocked out

within this (minimal) time period, SDN control messaging based on SF protocols can be sliced in time so that it does not interfere with other network traffic.

[Q3]: *Are there solutions to reduce SDN overhead?* The conclusions drawn from Section 2.1, outline that minimal SDN functionality must cater for *collection*, *configuration*, and *reaction* services. Not only can all three of these serves be facilitated through SF-based protocols, but the *one-to-all* nature of flooding means that it is ideally suited to SDN *configuration*, allowing a central controller to configure all nodes within the mesh in a single flooding period.

[Q4]: *Can SDN scale in a low-power wireless mesh?* This section has identified that the issue of SF scalability is an open research question. However SF scalability is concerned with scaling to hundreds of nodes; works covered in Section 2.1 highlight that SDN has scalability issues in far smaller mesh networks, consisting of only tens of nodes. By this measure, a SDN solution based on SF protocols could provide scalability within a local cluster of nodes within a larger mesh network.

2.4 A Review of SDN in Low-Power Wireless Mesh Networks

Section 2.1 reviewed the general concept of Software Defined Networking (SDN), showing how the *forwarding*, *network state*, and *configuration* abstractions underpin the SDN architecture and provide a platform that allows network intelligence to be logically centralised, and elevated away from individual hardware devices onto a single unified control plane.

Recent research has considered how to extend SDN control to low-power wireless networks. Whereas traditional SDN concepts have been successfully applied to other networking environments, such as data centres and optical [4], the constraints of low-power wireless networks, as covered in Section 2.2, pose considerable challenges to centralised control architectures.

Acronym	Definition
WMN	Wireless Mesh Network
WSDN	Wireless Software Defined Network
SDWN	Software Defined Wireless Network
SDWSN	Software Defined Wireless Sensor Network
SDWMN	Software Defined Wireless Mesh Network

Table 2.7: SDN in low-power wireless acronyms used by the works outlined in this section.

This section therefore provides a comprehensive review of recent literature exploring SDN in wireless mesh networks, with particular emphasis on IEEE 802.15.4, it draws concepts and discussion from surveys detailing this research area [20, 21, 23], and highlights how they attempt to overcome the challenges of implementing SDN within a constrained environment. Figure 2.8 shows a high level overview of this literature, sorting by *proposal*, *simulation*, or publicly available *implementation*. As many of the works define their own acronym to describe the concept of SDN in mesh and low-power wireless, these are covered in Table 2.7.

Table 2.8: Summary of SDN proposals, simulations, and implementations for wireless mesh and low-power wireless networks.

Publication	**Year**	**Contribution**	**Category**	**Description**
Sensor OpenFlow [103]	2012	Proposal	Data Plane	Mechanisms for reducing SDN control overhead in SDN-enabled WSNs.
SDWN [105]	2012	Proposal	Data Plane	Data aggregation and routing, serving as the basis for SDN-WISE.
SDN-WISE [26] [2]	2015	Implementation	Data Plane	Uses stateful flowtables to return intelligence to the mesh and reduce control overhead.
SD-WMN [106]	2016	Proposal	Data Plane	Spectrum allocation and scheduling.
OpenFlow for Mesh [25]	2011	Proposal	Control Overhead	OpenFlow control delay and overhead in a wireless mesh .
Cognitive-SDWSN [107]	2015	Simulation	Control Overhead	Reinforcement learning to improve energy efficiency by reducing SDN overhead.
IT-SDN [27] [3]	2017	Implementation	Control Overhead	Builds on TinySDN, adding separation between controller, neighbour, and southbound protocols.
Coral-SDN [108, 109] [4]	2017	Implementation	Control Overhead	Dynamically configures timers to reduce RPL control messages and improve SDN scalability.
Trickle in SDWSNs [110]	2018	Simulation	Control Overhead	Implements trickle timers to gradually reduce the frequency of SDN-WISE control messages.
Smart [111]	2014	Proposal	Controller Placement	Resource allocation and management through a centralised WSN controller.
TinySDN [112] [5]	2015	Implementation	Controller Placement	Provision for multiple controllers in a wireless mesh.
Spotled [113]	2016	Simulation	Controller Placement	Hierarchical control architecture for software-defined WSNs.
SDSense [114]	2019	Simulation	Controller Placement	Decomposition of network functions and maximising network utilisation.
FTS-SDN [115]	2018	Simulation	TSCH and 6TiSCH	Extends SDN-WISE to the TSCH domain, providing SDN-based TSCH forwarding and scheduling.
Whisper [116] [6]	2018	Implementation	TSCH and 6TiSCH	Manipulates 6TiSCH networks through spoofing 6top commands from 'Whisper' nodes.

[2] https://github.com/sdnwiselab
[3] http://www.larc.usp.br/users/cbmargi/www/it-sdn
[4] https://github.com/SWNRG/coral-sdn
[5] https://github.com/TinySDN
[6] https://github.com/imec-idlab/whisper-repository

2.4.1 Summary of Literature

2.4.1.1 Data Plane

The authors of Sensor OpenFlow [103] identify SDN as a means of solving WSN resource under-utilisation, rigid policy based on application needs rather than current link and node requirements, and providing a network management framework. However, they identify many challenges faced by SDN in constrained environments. Crucially, they provide a template to allow OpenFlow flowtables to handle compact addressing schemes in non-IP WSNs, and propose the addition of new OpenFlow forwarding rules, such as data-aggregation. Additionally they explore the difficulties of implementing out-of-band control plane communication within a constrained network, citing excessive overhead from the OpenFlow protocol in a wireless mesh, and propose mitigating this through Control Message Quenching (CMQ) [104], whereby retransmissions of SDN control messages from individual nodes are throttled in order to give the controller time to respond. This ensures nodes don't repeat control signalling when a controller response is forthcoming.

SDWN (Software Defined Wireless Networks) [105] addressed the constraints of IEEE 802.15.4 on the SDN architectural model, and served as the basis for the publicly available, and highly-cited, SDN-WISE [26] implementation. Highlighting the numerous constraints faced in IEEE 802.15.4 mesh networks, the authors suggested that fundamental aspects of the OpenFlow approach are incompatible with the complex and varied requirements of low-power wireless, and advocated a fundamental rethink of how to achieve SDN in such an environment. Specifically, SDWN introduces a lightweight southbound protocol tailored to the limited IEEE 802.15.4 MTU, as well as a highly configurable flowtable that accounts for the limited memory available in low-cost, low-power hardware. Not only does SDWN provide a template for performing in-network data aggregation through the SDN flowtable, but the authors proposed a form of Protocol Oblivious Forwarding (POF) [35], allowing flowtables to match on a *byte index* and *length* within the packet, rather than following the OpenFlow approach of needing to define new rules for specific header fields.

The authors of SDWN furthered their initial work through the publicly available SDN-WISE [26] implementation. SDN-WISE demonstrates a *stateful* approach to SDN, similar to the one taken by OpenState [39] in wired SDN networks. Through the introduction of *state* within the SDN flowtable, SDN-WISE attempts to reduce controller overhead with the addition of node state in the flowtable, allowing varying actions to be performed with respect to that state (essentially turning the nodes into FSMs). Additionally, the authors define *OpenPath* packets that allow the creation of *paths* through the WSN by setting Flowtable forwarding entries at each hop along a route. However, although the SDN-WISE data plane is fully configurable through the SDWN flowtable, meaning anything treated as a 'data' packet can be forwarded, dropped, or manipulated according to installed rules, the control signalling imposed by SDWN is highly static. The decision to use the RIME [117] communications stack necessitated that basic topology

and controller discovery protocols had to be designed from scratch, rather than relying on mature protocols such as RPL [24]. Consequently, these simple discovery and maintenance processes compete for communications resources alongside the actual southbound SDN control protocol. This can result in high levels of contention (depending on statically set periodicity), and severely affects scalability.

Huang et al. [106] address the question of how the SDN *forwarding* abstraction should be approached in a IEEE 802.11 wireless mesh network. Considering utilisation of wireless spectrum, the authors argue that a centralised SDN controller should be able to provide optimal allocation of this resource across routing nodes, rather than the decentralised fashion taken by traditional approaches. However, the authors recognise that the centralisation of knowledge at a single point within the mesh leads to frequent signalling between SDN nodes and the controller that can lead to poor network performance. As such, the authors propose spectrum slicing to guarantee resources for out-of-band control traffic. This furthers the SDN concept of separation between data and control planes, extending this separation to the wireless spectrum itself.

2.4.1.2 Control Overhead

The authors of [25] examined the general performance of OpenFlow in IEEE 802.11 wireless mesh networks, while demonstrating SDN-based fast handover performance and mobility management. They showed that there can be delay in the time to set up a new rule from an OpenFlow controller to the forwarding devices, restricting the ability of the network to react swiftly to network changes (as identified in Section 2.1.7) where network devices need to communicate with the SDN controller in a time-critical manner. Additionally, they identified that the control signalling generated by OpenFlow is magnified in a wireless mesh, as control messages are transmitted multiple times along an end-to-end multi-hop route. Both of these aspects affect SDN scalability, however they determine that as long as the rule installation rate is low, then the extra overhead can be absorbed by the IEEE 802.11 mesh. Although not specifically focusing on low-power wireless, their findings on the effects of centralised SDN control within a wireless mesh formed the basis for some of the works listed in this section.

The authors of Cognitive-SDWSN [107] identify the issue of excessive SDN control and data plane signalling causing increased energy consumption and contention in a low-power wireless mesh. They present reinforcement learning mechanisms to perform optimal load balancing and resource utilisation across the mesh, in order to apply optimal flowtable rules that curb the signalling overhead, and build this approach on the Sensor OpenFlow architecture [103].

Based on the publicly available Tiny-SDN [112], the authors of IT-SDN [27] update the architecture to further ideas of abstraction and separation, drawing clear distinctions between the *southbound controller*, *neighbour discovery*, and *controller discovery* protocols. The authors identify serious issues in previous approaches to flow installation, highlighting that in traditional OpenFlow routing (as shown previously in Figure 2.4) will result in 15 separate transmissions

across a 4 hop path, and proposing both 'source routed' and 'multiple flow' approaches in order to reduce this overhead. However, both these approaches rely on global network knowledge at the controller, with an up-to-date and stable network state. While IT-SDN defines interfaces for both neighbour and controller discovery, it fails to specify the protocols that might provide this functionality. Additionally, the authors perform simulation and experimentation on a 15 and 5 node testbed respectively, raising questions of IT-SDN's scalability.

CORAL-SDN [108, 109] reduces the effect of overhead generated by other control protocols on the SDN stack, and uses a mechanism to reduce RPL control messages in a IPv6 based IEEE 802.15.4 network as nodes initialise and associate with the SDN controller. This frees up resources for the SDN protocol, improving its scalability. However reducing the frequency of RPL control messages may cause issues when trying to maintain end-to-end links between the controller and the edges of the network, particularly in interfered or dynamic networks.

Similarly, Hieu et al. [110] consider the trickle timer [118], used in RPL [24] to reduce the frequency of control messages as the network topology stabilises, however they approach the issue of increased energy usage SDN-WISE resulting from excessive control signalling (as SDN-WISE control messages are generated periodically) and compare this approach with a traditional 6LoWPAN solution. By applying trickle to reduce control messages as the network stabilises over time, they show that energy consumption in a SDN-WISE node can be reduced to equivalency with a 6LoWPAN node.

2.4.1.3 Controller Placement

The authors of Smart [111] propose central placement of the SDN controller at a base-station (root node) within the mesh network, outlining a layered mapping model for abstracting the mesh state. Although the authors presented an architecture rather than experimental findings, they highlighted some of the key challenges faced in the placement of control intelligence within a wireless mesh: such as whether control should be centralised or distributed, and how the placement of this intelligence affects energy consumption in nodes.

TinySDN [112] is a multi-controller SDN architecture built on the constrained OS for low-power devices, TinyOS [119]. In their approach, the authors explore deploying multiple controllers and attempt to reduce the latency of control communications by designating WSN sink nodes to act as SDN endpoints and host controller applications. SDN-enabled nodes in the network are then able to join their nearest controller without having knowledge of the multiple controller instances. Yet this initial work doesn't provide a solution for east/west controller communication, or maintaining a logically centralised control plane across the multiple distributed controllers. Additionally there are questions regarding the scalability of the solution given that simulations were run on a 7 node network.

The authors of TinySDN extend their platform to incorporate a two-tier hierarchical controller architecture in Spotled [113]. Identifying the increased overhead resulting from the centralisation

of control in the wireless mesh, they propose delegating some intelligence back into the mesh. Furthermore they highlight the issue of increased signalling when a SDN node first joins the network, showing how controller communication necessary to install flowtable rules introduces initial delay and network instability. To address these issues, Spotled implements a hierarchical controller model where local controllers are responsible for managing SDN sensor nodes in its immediate area, while a global controller maintains the overall network state and synchronisation between the local controllers (addressing a weakness in their earlier work). While the overall network state remains logically centralised, the local controllers allow fast network association and installation of flowtable rules.

SDSense [114] similarly defines a hierarchical controller framework, and proposes the decomposition of network components based on *slow* and *fast* requirements, as well as deriving an an algorithm to optimise Time Division Multiple Access (TDMA) resource utilisation across the network. A logically centralised controller manages topology-control, TDMA scheduling, and baseline data-rate modules to allocate static or *slow* network components from a global perspective, while congestion control and data-rate reallocation modules react to the state of local controllers employed on each SDSense node. Through simulation and modelling, the authors demonstrate dynamic network reconfiguration, and show significant improvements in network performance over other solutions, however don't provide a full implementation for comparison against other SDN solutions.

2.4.1.4 TSCH and 6TiSCH

The authors of SDN-WISE extend their framework to the IEEE 802.15.4e TSCH MAC layer for industrial wireless in FTS-SDN [115] (Forwarding and TSCH Scheduling over SDN), and present an SDN-based solution for handling topology changes (such as node mobility) within a TSCH network. They conclude that mobility relies on a topology update period grater than that of topology changes, and demonstrate successful handover of the mobile node in a 5 node network simulation. Although the potential introduction of SDN configurability and abstraction to IEEE 802.15.4 TSCH is an interesting prospect, the authors don't cover these aspects, while the handover mechanisms introduced in this work are covered more extensively in the IETF 6TiSCH standard [2]. Furthermore, the 5 node testbed used in this study is too small to be accurately representative of the large-scale industrial networks that IEEE 802.15.4e TSCH aimed to address.

Whisper [116] re-evaluates the role of SDN in low-power wireless with respect to IEEE 802.15.4e TSCH, and the recent forwarding, scheduling, and routing mechanisms introduced in 6TiSCH. The paper recognises that the move to a TSCH-based MAC layer introduces additional complexities on top of the existing challenges facing SDN within a wireless mesh. Specifically, the issue of additional overhead becomes more acute, as duty-cycling restrictions and scheduling mean that in-band control paths without sufficiently allocated resources will suffer from large

delays. The authors propose strategically placing Whisper nodes to manipulate the network using existing RPL [24] and 6TiSCH control signalling (6top Protocol (6P) [120]). A central controller translates high-level control policies into a limited number of control messages, and can then delegate control functions to Whisper nodes. These nodes are then able to discretely affect their 1-hop area through the targeted injection of RPL and 6P packets. This work addresses the issue of excessive control overhead detrimentally affecting the performance of SDN controlled (or a centrally scheduled) 6TiSCH mesh; however, though the mechanism of injecting control signalling into the surrounding network is a highly novel approach, it raises some security concerns around the practicality of spoofing 6TiSCH control messages.

2.4.2 Conclusions

From this overview of relevant literature exploring the application of SDN in low-power wireless mesh, conclusions are drawn with reference to Research Questions [Q1, Q3, Q5]:

[Q1]: *What are the fundamental features of a minimal SDN solution?* The authors of SDWN [105] provided a complete proposal for SDN in IEEE 802.15.4 low-power wireless mesh. This architecture was followed up by one of the few publicly available implementations, SDN-WISE [26]. As such, the SDWN approach became a standard model and inspiration for much of the literature in this area, including elements of the μSDN [C1, C2] architecture presented in Chapter 3. Additionally, other works referenced in this section (such as TinySDN [112], SDSense [114], and Whisper [116]) challenge the belief of a logically centralised controller, and advocate a distributed intelligence model to account for dynamic topology changes and push intelligence closer to the edge. Between them, these works build an argument that SDN in IEEE 802.15.4 low-power wireless mesh must diverge from the traditional SDN model; specifically:

- Flowtable rules shouldn't be tied to the southbound protocol (e.g. it should follow an index/length matching approach rather than matching on specific header fields).
- New action types, such as data-aggregation, are critical to supporting SDN in the low-power wireless mesh.
- Decisions on the placement of control nodes within the mesh should be fluid, allowing intelligence to be pushed to where (and when) it is needed.

[Q3]: *Are there solutions to reduce SDN overhead?* and **[Q4]:** *Can SDN scale in a low-power wireless mesh?* It is notable that much of the literature covered in this section identifies, with varying emphasis, that the primary barrier to implementing SDN in a low-power wireless mesh lies in the volume and latency of control signalling. Not only is this overhead a matter of providing timely control decisions and an updated network state at the controller, but it directly affects the scalability of a SDN-enabled network. If the result of additional SDN control traffic is that the size of the network can't scale beyond a handful of nodes, then the benefits of SDN can't be

applied to real-world industrial mesh networks, in which a single mesh may need to support over a thousand nodes. Across these works, a number of proposals have been put forward to try and reduce this control overhead, including:

- Control Message Quenching (CMQ) [103, 104].
- Data aggregation and in-network packet processing [105].
- Flowtables configured as state machines [26].
- Distributed controller systems [112–114, 116].
- Resource optimisation through reinforcement learning [107]
- Manipulation of existing control signalling [108, 116]
- Novel flow-installation approaches [26, 27]

2.5 Summary and Conclusions

This chapter has provided context for the technical content presented in in this Thesis. Not only has it covered background material on SDN and low-power wireless networks, relevant to Chapters 3 and 4, but it has introduced recent research exploring the use of CT and SF to achieve highly reliable and ultra low-latency communications in extremely constrained mesh networks, which underpins the work in Chapter 5. Finally, the works reviewed in Section 2.4 establish that when applying SDN concepts to constrained low-power wireless mesh networks, the issue of control signalling between the individual nodes and the SDN controller becomes a fundamental hurdle. However, the benefits of an 'softwareized' IoT network, capable of dynamically supporting the requirements of multiple tenants, are evident. The challenge lies in how to adapt SDN architecture to constrained environments in a scalable manner, without sacrificing the reconfigurability and agility that SDN brought to traditional networks.

CHAPTER 3

μSDN: A LIGHTWEIGHT SDN ARCHITECTURE FOR THE INTERNET OF THINGS

The previous chapter examined the fundamentals of Software Defined Networking (SDN), and provided background information on the low-power IEEE 802.15.4-2015 'IoT Stack'. Furthermore, it delivered a comprehensive review of current and recent works exploring the application of SDN concepts within a low-power wireless network, and how to approach the considerable challenges of such a constrained environment.

This chapter expands on these challenges, with specific focus on IEEE 802.15.4-2015 multi-hop mesh networks employing asynchronous MAC layers[1] (such as CSMA/CA, or duty-cycled approaches such as ContikiMAC) [121]. With reference to current works, and conclusions from Chapters 1 and 2, Section 3.1 firstly argues the case for a new SDN architecture for low-power wireless mesh networks, while Section 3.2 explores design considerations arising from this. This provides motivation for the design and implementation of μSDN, an embedded SDN architecture for Contiki OS [122]. Architectural and operational aspects of μSDN are then covered in Section 3.3, showing how it achieves full interoperability with 6LoWPAN and RPL, and how it incorporates a number of mechanisms to reduce control signalling between mesh nodes and the embedded SDN controller - implemented as part of μSDN. Section 3.4 evaluates μSDN performance on emulated hardware (TI MSP430F5438 CPU and CC2420 radio) in the Contiki Cooja simulator, and explores a use-case demonstrating how μSDN is capable of slicing the network on a per-flow basis to provision critical flows with alternate forwarding paths in order to avoid localised network interference.

Material in this chapter is supported by publication [C1], which argues that interoperability with existing topology discovery and maintenance protocols is key to providing a scalable and

[1] For extension of μSDN to 6TiSCH and the IEEE 802.15.4e TSCH MAC, the reader should be directed to Chapter 4.

robust Southbound (SB) link between the SDN controller and the mesh network. The reader should note that while this chapter solely considers this publication, which focuses exclusively on IEEE 802.15.4 asynchronous MAC layers (such as CSMA/CA), the exploration of SDN control issues in a 6TiSCH network is later covered in Chapter 4. μSDN represents a considerable coding effort over two years of development: consisting of multiple packages, dozens of files, and many thousands of lines of code. It has subsequently been released as a publicly available repository on GitHub [R1], where it is regularly cloned has resulted in a number of forks. This repository provides a platform for academic research and experimentation into SDN for low-power wireless network, and has recently been used to develop an energy-aware SDN/NFV framework for IoT [C4].

Results in this chapter support research questions [Q1, Q2, Q3, Q5], and it makes the following specific contributions:

- Minimal functionality and a lexicon for SB controller communications is established.
- Coexistence with the RPL routing protocol is proposed to facilitate SDN controller discovery and maintain control links to mesh nodes.
- μSDN is implemented, a lightweight[2] SDN architecture for IEEE 802.15.4-2015 6LoWPAN networks.
- Reduction of SDN control overhead is explored though the implementation of a number of mechanisms in μSDN.
- *Atom* is implemented, an embedded controller for μSDN, designed to reduce delay for SDN control decisions within the mesh.
- Simulations are performed showing that μSDN exhibits minimal additional overhead in comparison to current 6LoWPAN + RPL networks.
- A use-case is presented, demonstrating how μSDN can be used to install rules that allow nodes to take alternate routes around localised external mesh interference.

3.1 The Case for a Low-Power Wireless SDN Architecture

Over the past decade, a prevailing trend in networks research has been the drive towards network 'softwarization', and the associated opportunities that a programmable network infrastructure can bring. Chapter 2 introduced how SDN, defined as the separation of control and data planes through multi-layered abstractions, has been the most visible and well-publicised aspect of this idea. Fundamentally, these abstractions form the basis of a Network Operating System (NOS),

[2]These thesis considers μSDN lightweight in terms of required control overhead and device memory usage - in comparison to traditional wired SDN architectures.

which in turn can support both network and network function virtualisation. As industrial and commercial value has drifted away from locally hosted applications to remote platforms, these technologies have been key to enabling the vast cloud and data-centre infrastructures that support the current crop of internet services. While virtualisation allows network resources to be provisioned and torn down in response to application needs, SDN provides a means to dynamically configure and programme those resources while abstracting the underlying complexity away from network functions.

As was also discussed in those previous chapters, a significant proportion of future global traffic growth will be driven by IoT networks [10]. The use-cases for such networks are widespread and diverse, with potential to disrupt, enhance, and create markets where previously none existed. While the traditional approach to IoT has involved the deployment of data collection applications on top of Wireless Sensor Networks (WSNs), there is a commercial interest in maximising the value of the mesh through dynamic configuration of resources in order to support multi-application/tenant slicing and guarantees; indeed, this goal is a prime motivator for the industrial sponsor of this PhD.

Section 2.4 therefore reviewed current and previous works exploring the application of SDN in IEEE 802.15.4 low-power wireless mesh networks. Yet while a number of the works provide publicly available repositories [26, 27, 108, 112, 116], at the start of this PhD the only available implementation was SDN-WISE [26]. Although SDN-WISE (based on the earlier SDWN [105] architecture) provides a full solution for academic research on SDN in a constrained mesh environment, a number of weaknesses supported the argument for establishing an alternate approach, namely:

- Lack of support for IPv6 through 6LoWPAN [84] addressing, header compression, and fragmentation.
- The authors' implement their own topology discovery and maintenance approach rather than adopting well-known and tested alternatives, such as RPL [24].
- Control links between the mesh and SDN controller are maintained in the SDN flowtables. This introduces circular dependency as nodes require a reliable link to the controller in order to instruct them on how to establish a link to the controller. For example, if the control link becomes unreliable through topology or environmental changes.
- Control signalling is generated by timers set with fixed values, causing possible contention depending on the periodicity of neighbouring nodes (although recent works explore the introduction of the trickle timer to SDN-WISE [110]).
- Control decisions are made outside the mesh on a controller sitting on the backbone network, increasing the distance and delay between mesh nodes and control intelligence.

Crucially, SDN-WISE's reliance on the RIME [117] communications stack for Contiki, which provides no networking layer support and leaves this to the implementation, severely constrains its scope for interoperability with low-power Industrial IoT (IIoT) networks: which almost exclusively employ some variant of the IEEE 802.15.4 'IoT Stack' [2, 71–74] outlined in Section 2.2.2. These considerations provide motivation for the design and implementation of μSDN, a low-overhead SDN solution for IEEE 802.15.4 mesh networks. μSDN boasts interoperability with both 6LoWPAN and RPL, while introducing a number of mechanisms to reduce the frequency of control messages and signalling between mesh nodes and the SDN controller.

3.2 Design Considerations

Section 2.2 in Chapter 2 outlined how much of the complexity in applying SDN control within low-power wireless networks stems from the difficulty in communicating over half-duplex radio links across a multi-hop mesh topology. Without an Out-Of-Band (OOB) control link between network devices and the SDN controller, all SDN control messages must share these lossy links with other network communications (such as application data, neighbour discovery, and topology maintenance). With multiple services vying for this resource, managing access to it becomes a fundamental issue when considering the overhead generated by Southbound (SB) SDN control signalling. Before detailing architectural aspects of μSDN (the low-overhead SDN solution covered in this chapter) it is important to consider not only how this issue affects communications between mesh devices and the SDN controller, but also how some of the commonly held assumptions from traditional SDN approaches (outlined in Section 2.1) conflict with the limitations and constraints inherent in a low-power wireless mesh (Sections 2.2.1 and 2.2.2).

This section subsequently highlights the design considerations that were taken during the implementation of μSDN, stemming from the particular and considerable challenges faced by centralised SDN control architectures in low-power wireless networks.

3.2.1 Device Hardware Limitations

Over the past decade SDN research has primarily targeted networks comprising of powerful hardware switches, data stores, and controllers: able to support large flowtables, rapidly service thousands of requests, and maintain complex network states.

SDN switches have the capacity to store, and quickly sort through, many thousands of entries within a single flowtable, and can be dynamically reconfigured to perform a multitude of network functions. On the control plane, SDN controllers can manage a highly complex and distributed view of all network devices, run diverse and computationally expensive network applications, and maintain concurrent virtual networks.

Yet these capabilities aren't readily transferable in the low-power wireless mesh solutions that have proven themselves the mainstay of IIoT. Cheap, Commercial-Off-The-Shelf (COTS)

hardware, paired with ultra-low energy consumption, delivers a cost-effective solution in areas such as smart-metering and industrial process monitoring, where devices may need to operate over long periods with minimal power. Consequently, these networks consist of extremely constrained devices with limited energy, memory, and processing capabilities; however, these restrictions allow devices to operate for months, or even years.

Unfortunately, this is particularly limiting for SDN, where its reliance on powerful hardware solutions cannot be replicated or matched with in a low-power wireless environment. Whereas typical SDN switches can support thousands of flowtable entries using high-speed Ternary Content Addressable Memory (TCAM), and provide MBs of memory for packet buffering, low-power wireless System-on-Chip (SOC) devices typically provide tens of KBs of Read Only Memory (ROM) and just a few KBs of Random Access Memory (RAM). Additionally, slow CPU clock speeds of just a few MHz severely limit the capability of individual nodes to quickly perform complex calculations.

3.2.2 Sensitivity to External Interference

Not only do low-power requirements dictate the use of constrained device hardware, but the need to conserve energy permeates all layers of the network stack. Concessions are made at the IEEE 802.15.4 PHY and MAC layers to support this, and the low-power nature of transmissions means the radio links are characteristically lossy. This can result in sensitivity to interference from nearby higher-power communications operating at the same frequency: particularly at the 2.4GHz unlicensed bands where both IEEE 802.11 WiFi [123] and IEEE 802.15.1 Bluetooth [12] also operate. Potentially, this can affect entire branches of the network, and hamper the delivery of messages from/to sensors and actuators. In a multi-hop SDN network relying on centralised control, interference at any point along the control link could prevent nodes from querying or receiving instructions from the controller, rendering them effectively orphaned from the network in a similar manner to the scenario presented in Figure 2.7 from Chapter 2.

3.2.3 Packet Forwarding and Fragmentation:

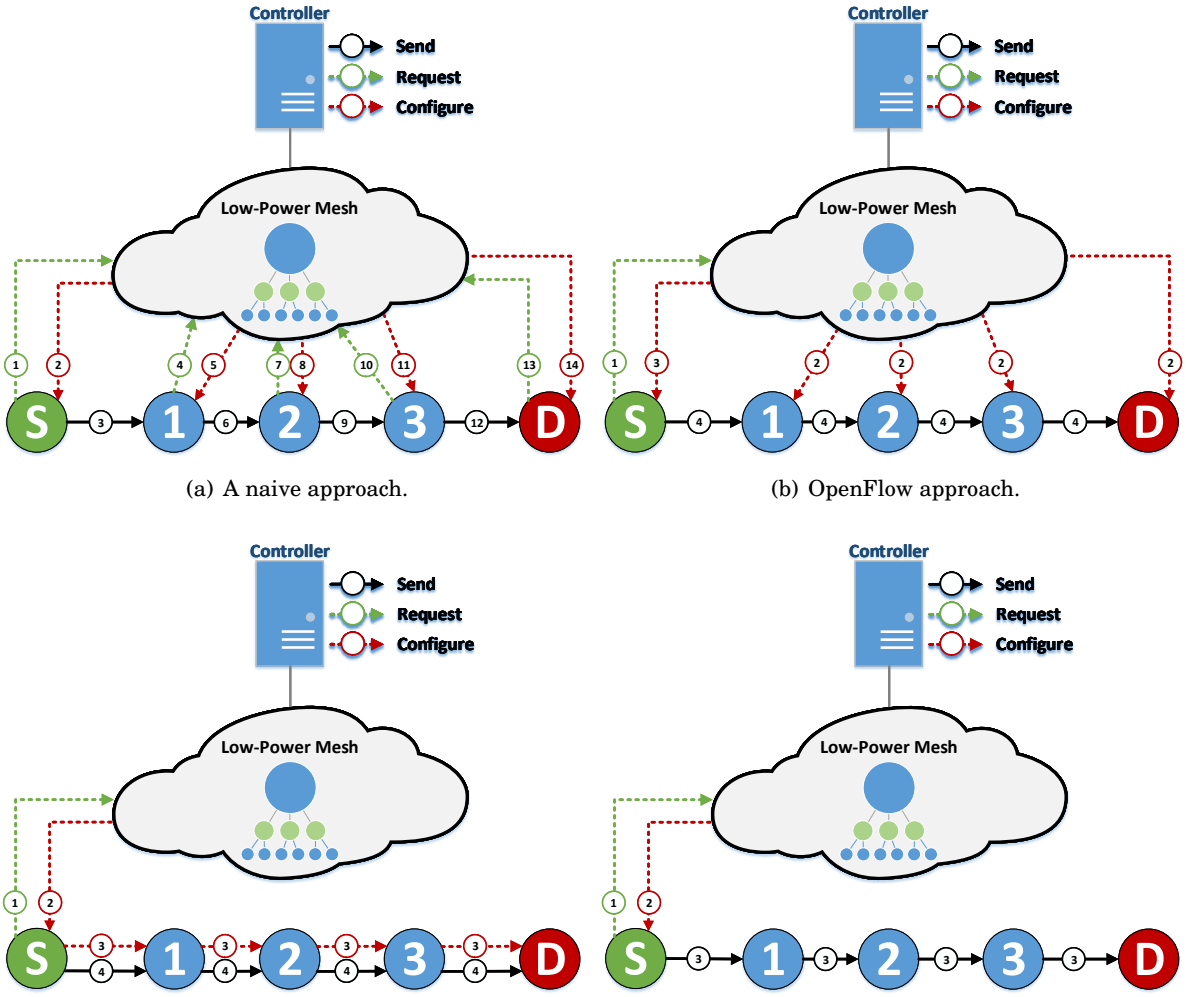
Section 2.2.2 presented the IEEE 802.15.4 frame structure, and showed how the standard employs a Maximum Transmission Unit (MTU) of 127B in order to reduce the maximum data lost in the event of packet drops. After the MAC Protocol Data Unit (MPDU) header, 6LoWPAN [84] then introduces header compression and fragmentation mechanisms that provide support for IP connectivity across low-power mesh networks. However, even though 6LoWPAN frees up some of the space taken by the IPv6 header, the space allocated for payload data is still restricted in comparison to that needed by traditional SDN OpenFlow messaging.

Although larger packets can be fragmented, transmitted, and then reconstructed on a hop-by-hop basis (with recent work exploring reconstitution at the destination [88]), additional transmissions would cause unnecessary delay to SDN control signalling. Therefore, in order to

prevent 6LoWPAN packet fragmentation and hence multiple transmissions per packet, SB SDN control messages need to fit within this allotted length.

3.2.4 Approaches to Multi-Hop Southbound Communication

Many of the works reviewed in Section 2.4 highlighted the challenge of applying centralised control signalling across a multi-hop mesh. Each hop that's required in order to reach the destination node adds an additional communication which, in turn, depletes node energy, uses up scarce spectrum resources, and increases end-to-end delay. Considering a SDN forwarding scenario, like the OpenFlow routing example provided in Figure 2.4 from Chapter 2, Figure 3.1 provides a graphical summary of various approaches to SDN flow installation in a low-power wireless mesh.



(a) A naive approach.

(b) OpenFlow approach.

(c) SDN-WISE approach.

(d) μSDN approach.

Figure 3.1: Summary of SDN flow instantiation approaches within a low-power mesh network.

Four approaches are identified, with each approach further reducing the control overhead necessary to form a route between nodes $S \rightarrow D$. These communications are broken down below, detailing the number of multi-hop and single-hop communications required in each scenario.

a) *Naive approach (x10 multi-hop, x4 single-hop)*: Each node along the path must individually request instruction from the controller. Once all nodes have successfully communicated with the controller, the flow is established.

b) *OpenFlow approach (x6 multi-hop, x4 single-hop)*: Once the initial node requests the flow, the controller will install flowtable entries at each node along the path, using the link between each individual node and the controller.

c) *SDN-WISE approach (x2 multi-hop, x8 single-hop)*: Once the initial node requests the flow, the controller embeds the path between S and D in a packet that installs the flow at every node along the path. Any future packets sent in that flow are routed according to the flowtable entries installed on each node.

d) *μSDN approach (x2 multi-hop, x4 single-hop)*: Once the initial node requests the flow, the controller installs the path as a flowtable action on S. Any future packets in that flow are then source-routed from S.

It is clear from these four approaches that the μSDN approach in Figure 3.1d results in the least control overhead, albeit at the expense of the fine-grained configurability provided through the naive approach in Figure 3.1a: i.e. when the controller installs the full path at S, there is no way of knowing if the links along that path are still active, and it must trust that the routing policies in place have taken this uncertainty into account.

3.2.5 Maintaining Control Links in a Multi-Hop Mesh

As discussed in Chapter 2, control traffic generated by SDN is the result of one of three processes: *collecting* network state information, *configuring* devices, or *reacting* to new data flows. Servicing these control processes requires the network to provide reliable, low-latency links between network devices and the controller: allowing the SDN architecture to maintain an up-to-date network view and rapidly and dynamically apply application logic. The ability to be able to quickly convey information and decision making between the control and date planes is therefore vital within an SDN network.

However, relying on a logically centralised authority for control decisions means that SDN architecture can potentially burden the network with a high associated overhead. When considering traditional wired or optical networks, this overhead can be problematic but is ultimately manageable thanks to reliable, low-latency control links and sufficiently powerful hardware. Furthermore, dedicated Out-Of-Band (OOB) control links can allow traditional SDN architectures

to concurrently configure multiple devices on the network without affecting application traffic. Yet even without the OOB channels, traditional SDN networks can still achieve highly reliable, low-latency links between the data and control planes, and OpenFlow SDN switches are capable of querying their controller within a few hundred of microseconds [124].

Within a low-power mesh network this SB link between SDN controller and network devices becomes highly critical. End-to-end latency is aggregated over multiple hops, while uncertainty is compounded at each link. The shared nature of the underlying radio medium means that SDN messages must compete with other control signalling, which can cause further delay and jitter, while fading and external interference from other devices can severely affect a channel. Relying solely on a centralised authority to maintain the links between itself and the mesh can result in the unfortunate situation where a node is incapable of querying the controller in order to repair its own control link.

In a multi-hop mesh network, control links *must* be maintained through either a distributed routing protocol or flooded communications. While a simple topology discovery approach like that taken in SDN-WISE [26] allows nodes to maintain their control links based on hop distance, existing distributed routing protocols like RPL [24] provide proven and robust mechanisms for maintaining mesh integrity.

3.2.6 Summary of Design Considerations

If the standard model of SDN architecture found in traditional SDN deployments is supported by the two pillars of *reliable and low-latency control links* paired with *sufficiently powerful hardware*, as has been explored in this section, then a number of considerations and trade-offs must be made when applying SDN architecture within the constraints of low-power wireless networks. The architectural requirements and restrictions arising from these trade-offs are broken down into four core areas: *Control Plane*, *Data Plane*, *Distributed Network State*, and the *SDN Controller*. These requirements form the basic objectives and motivation behind decisions made in the next section, which details the design and implementation of μSDN, a low-overhead SDN stack for low-power wireless networks.

Control Plane:

- *Eliminate Fragmentation* through tailoring the SDN control protocol so that it doesn't exceed the allocated packet size after the link layer and 6LoWPAN headers are subtracted from the MTU.
- *Reduce Packet Frequency* to minimise potential for congestion, as well as reducing potential retransmissions at the link layer
- *Match on Byte Arrays/Index* rather than specific header fields, allowing greater reconfigurability and programmability in the mesh.

Data Plane:

- *Throttle Control Messages* to ensure that repeated requests, from a node to the controller, are not sent in quick succession: for example, if a node receives unknown bursty data it may try to send multiple controller requests for the same flow. This throttling solution also has security implications that could present a possible defence against a Denial-of-Service style attack.
- *Refreshing Flowtable Timers* reduces reliance on instructions from the controller as repeated successful matches will not expire. This is, however, a trade-off between configurability and performance.
- *Assign Flow Priority* to ensure commonly used flowtable entries are checked first, reducing lookup delays.
- *Reuse Flowtable Matches/Actions* by eliminating repeated entries. For example, if there are entries for two flows which are then forwarded to the same destination, that forwarding action should be stored as a single item, rather than being included in both entries.

Distributed Network State:

- *Maintain a Distributed Control Topology*, supporting proven and robust distributed routing protocols, rather than relying on a central authority to maintain its own control links.
- *Use Source Routing* to prevent intermediate nodes from generating new control requests as the packet is transported from source to destination (assuming that the intermediate nodes have no rules for that flow).

SDN Controller:

- *Embed the SDN Controller* to allow basic control requests to be responded to more quickly, rather than sending them on a controller on the external IPv6 backbone network.

3.3 Implementing μSDN: a Lightweight SDN Stack for IoT

By addressing the design considerations in the previous section, μSDN maintains the core Software Defined Networking (SDN) concepts of data plane abstraction coupled with a centralised controller, whilst minimising SDN control signalling overhead. Sitting above the IPv6 Layer as shown in Figure 3.2, μSDN is integrated with the IEEE 802.15.4-2015 [1] protocol stack and is fully interoperable with legacy nodes, taking into account the compression and fragmentation mechanisms of 6LoWPAN [84], while employing RPL [24] to establish control links. Crucially, RPL provides a distributed means of controller discovery as well as robust maintenance of the multi-hop link to the controller, though this could be switched-out for other distributed routing

protocols. Furthermore, μSDN implements a lightweight SDN controller at the RPL DAG root: this pushes intelligence onto the mesh rather than solely facilitating control decisions through a border router.

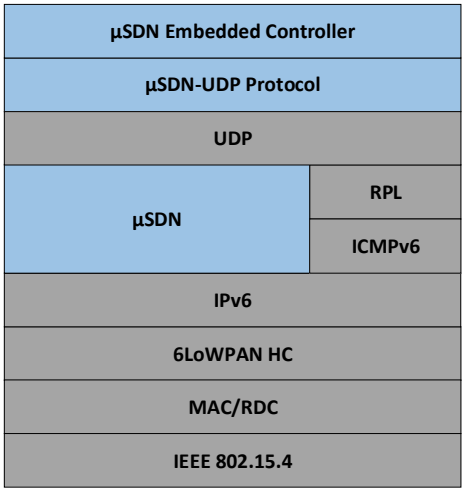


Figure 3.2: The μSDN stack. Blue denotes the μSDN layers, whilst grey shows the IEEE 802.15.4 PHY, MAC, networking, and transport layers.

This section discusses the implementation and architectural aspects of μSDN, a lightweight SDN framework built for Contiki OS [122]. Although other SDN implementations for low power wireless were made available during the later part of this PhD, μSDN was one of two concurrent works to first advocate full interoperability with 6LoWPAN and RPL [108, C2], and the first to explore SDN control link issues in a 6TiSCH [2] network (detailed in the Chapter 4). The primary motivation behind μSDN was to provide a fully functional SDN framework for IoT, whilst acknowledging and adapting to the limitations of low-power wireless. In this manner, μSDN contributes a lightweight Network Operating System (NOS) for constrained networks, provides a platform for multi-application/multi-tenant slicing, and helps support configurable Industrial IoT (IIoT) business models.

3.3.1 μSDN Architecture and Operation

μSDN builds on some of the concepts proposed in the recent works highlighted in Chapter 2, whilst considering the architectural requirements and trade-offs outlined in the previous section. Additionally, μSDN takes into account one of the key failings inherent in the traditional SDN approach for wired networks, which (contrary to the aim of SDN) often locked devices into specific OpenFlow versions, and sets out to provide an abstract framework on which many aspects can be extended or swapped out completely. Figure 3.3 highlights how μSDN employs a modular architecture designed around three layers, the *SDN Adaptor*, the *SDN Engine*, and the *SDN*

Driver. The later of which interfaces with a suite of components that provide core SDN functions: *Statistics*, *Flowtables*, *Signalling Reduction*, and *Controller Discovery*. These three layers, and the underlying SDN functional components, are implemented on all mesh devices, including the root node.

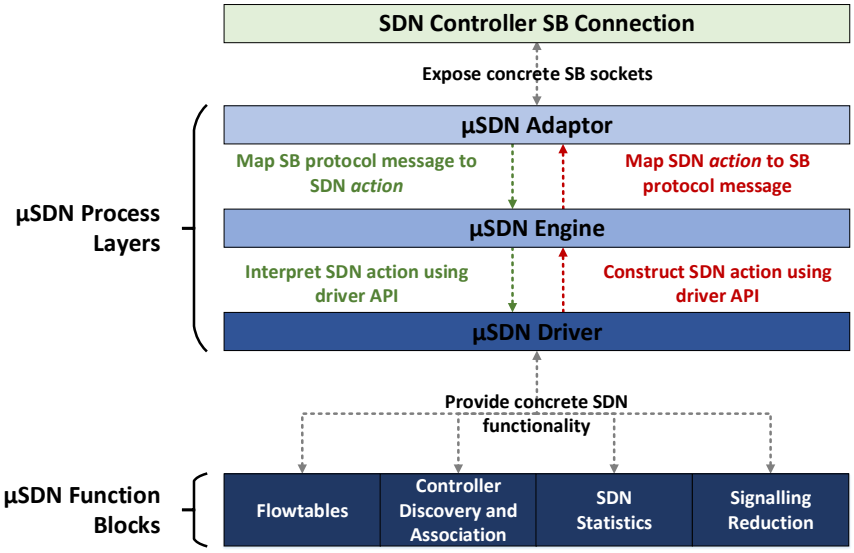


Figure 3.3: μSDN architecture and operation. μSDN operates on all mesh nodes, including the mesh root node hosting the μSDN-Atom controller.

Instructions to and from the controller are sent via the SDN Adaptor, which exposes a protocol-specific connection (such as OpenFlow, or in this case μSDN-UDP) to the SDN controller, and maps control messages into a μSDN *action*. This is then sent to the SDN Engine, which interprets the action and implements appropriate logic from core functional components through an abstract interface defined by the SDN Driver. Through this layered design fundamental features, such as flowtables and maintaining a link to the controller, can be separated from the specifics of the chosen SDN protocol logic: whether that's μSDN-UDP, or some other control protocol. Each of these components, and their operation, are outlined below.

SDN Adaptor: Discussed extensively in the initial chapters of this thesis, the fundamental concept underpinning SDN is that of separation between the data and control planes, and the abstraction layers that are consequently created. Unsurprisingly, control links need to be established between these planes, allowing a logically centralised control entity to program devices and configure the network through a Southbound (SB) protocol. Section 2.1 covered a number of SDN protocols that facilitate this communication, and SDN controller implementations often provide adaptors for a host of SB interfaces. Taking into account that there is no one 'killer' protocol for SDN communication and to support experimentation of different SB protocols, the SDN Adaptor provides an abstracted interface to the SDN Engine, whilst exposing a protocol-

specific socket to the SDN controller. SB protocols implemented at the adaptor are required to map their various messages to underlying *actions* defined by the engine. In this manner μSDN can connect to controllers that utilise different SB implementations, whilst leaving the underlying layers unchanged, and affords the flexibility to extend the μSDN by switching out the lightweight μSDN-UDP protocol for other protocol implementations.

SDN Actions: The concept of *actions* are defined in order to provide an abstraction between messaging from the SDN controller, which shouldn't necessarily need to be the μSDN-UDP protocol, and the specifics of how *actions* are handled within the engine itself. Table 3.1 shows how these commands provide a common language for a standard set of functions, follow the basic SDN services of *collection*, *configuration*, and *reaction*, and indicates the direction (*upwards/downwards*) of this traffic within the mesh topology (assuming a centralised control entity).

Table 3.1: SDN *actions* as defined by μSDN.

μSDN Action	**Service**	**Direction in Mesh**
Join an SDN controller	*collection*	↑
Request instruction from the Controller	*reaction*	↑
Update the controller with local node statistics	*collection*	↑
Update the node with a new SDN configuration	*configuration*	↓
Program the node's flowtable	*configuration/reaction*	↓

SDN Engine: The SDN Engine serves as an implementation-specific translator for the *actions* coming from the adaptor. It interprets how actions should be performed given the functionality exposed through the API provided by the SDN Driver, and it uses this functionality to achieve the desired intent of the generic instruction (or vice versa: using information from the primitives to construct an appropriate action to send to the adaptor). These functions may be something simple, such as forwarding one or more flows to a neighbouring node, or it could potentially be more complex: such as defining load balancing functions, which require a number of different flowtable entries to be installed; or deciding whether to join a new controller. While *actions* map SB protocol messages to SDN concepts in broad-strokes, the engine fine-tunes these into specific operations that can be performed by the SDN Driver through manipulation of the SDN flowtable, or the other SDN functional components.

SDN Driver: The SDN Driver presents a simplified and abstract interface to the engine, removing the complexities of how the functionality provided by underlying SDN components should be handled: such as the installation and removal of entries from the SDN flowtable, collection of statistics to be sent to the SDN controller, mechanisms to reduce the level of control signalling, or how to discover and maintain controller connections.

Flowtables: The initial sections of this chapter highlighted that while traditional SDN architectures are supported by powerful hardware switches able to support many thousands of rules, the hardware restrictions of low-powered wireless devices means that typically only tens of kilobytes of memory is available. This then needs to be shared with the rest of the IEEE 802.15.4 stack, further reducing a limited resource. Assuming that the flowtable can only hold, at most, a few dozen entries, the question becomes one of how best to utilise that available memory and still provide the configurability and flexibility that are the hallmark of an SDN architecture.

In order meet this challenge, μSDN extends the concept of Protocol Oblivious Forwarding (PoF) [35] to low-power devices. Instead of flowtable rules matching on specific packet header fields (like in the OpenFlow approach), both the *match* and *action* columns in the μSDN flowtable are structured as an *index*, *length* (in bytes), and boolean *operation*. Although still limited in size, adopting such a configurable mechanism makes μSDN flowtables extremely powerful, allowing the flowtable to be programmed with rules that can match on any field of any ingress packet, and manipulate that packet in any manner deemed necessary by the controller.

Yet when a packet needs to be parsed over even a handful of rules, hardware constraints again cause further issues. The power-hungry processing units and fast TCAM memory available in traditional switches ensures packets can be checked against thousands of entries within a few microseconds. However, μSDN flowtables are implemented as a linked-list on an embedded device, where every single table lookup can cause millisecond delays to a packet: a common problem in embedded programming. When accumulated across multiple hops these processing delays can become considerable; something that control signalling, in particular, can be sensitive to.

μSDN addresses this in two ways. Firstly, it provides a means for controllers to configure multiple flowtables with varying priority levels. This, for example, allows the controller to configure a *whitelist* which is processed before the main flowtable. Packets matched in this *whitelist* are then handed back to the regular Layer-3 processes with minimal interruption and, for instance, can be used to allow RPL control traffic to pass freely without hindrance. Secondly, individual rules in the flowtable can be assigned priority. By explicitly assigning priority in the controller can then ensure critical flows that need to be immediately forwarded are matched quickly without incurring delay.

Controller Discovery and Association: Discovering resources and maintaining links across a multi-hop mesh network is an immensely difficult challenge that isn't unique to SDN. However, the unavoidable fact that SDN nodes rely on these links in order to communicate with the controller, and exhibit any sort of intelligence, makes it particularly acute. Just as a cellphone is seen as 'bricked' without an internet connection, an SDN node without a controller connection is similarly defunct.

μSDN avoids this problem by providing integration and interoperability with existing distributed routing protocols, such as RPL. Although the controller is free to install its own control

links, nodes can fall back on distributed routing in the event that they lose this connection. While μSDN specifically uses RPL Non-Storing (NS), this could in theory be any implemented protocol.

Being able to fall back on distributed *default* routing therefore confers an element of robustness to controller connections within μSDN. As RPL is self-healing, it ensures nodes always have a mechanism to try and establish control links in the event of topology changes due to external interference, fading, or mobility. Although in an ideal scenario a controller should determine optimal control routes, in a low-power wireless mesh this approach isn't practical. While previous low-power wireless SDN architectures have implemented their own distributed topology discovery protocols [26] these provide only basic management mechanisms and, given the unreliability at the lower layers, can be sensitive to changes across the multi-hop control link.

In μSDN, the controller join process therefore employs both the underlying RPL topology as well as the μSDN-UDP protocol. When the controller receives a RPL DAO (destination advertisement) message that has been propagated upwards from a joining node it will send a μSDN Configuration (CONF) message in return, in addition to the RPL DAO-ACK. The joining node uses this CONF message as acknowledgement that it is connected to the controller, as well as providing it with an initial SDN configuration (periodicity of statistics updates, default flowtable settings, etc.).

Statistics: Even with a robust and low-latency link between the controller and SDN node, the controller is only as knowledgeable as the available data. Regular reporting of statistics are a vital part of any SDN implementation, providing the controller with a view of the network state from which it can make informed decisions. However, to maintain an up-to-date view of the entire multi-hop mesh would require nodes to report data every time a change was detected for a number of key indicators: such as neighbour information, energy levels, and link quality. SDN flowtables add additional complexity in that each individual row also generates its own statistics on the number of times it has matched, and the remaining time-to-live for that entry.

In an effort to reduce the volume of data sent upstream to the controller, μSDN statistics reporting uses configuration parameters provided by the controller in order to maintain an up-to-date view of only selected statistics that are of particular and current interest. This ensures that only metrics relevant to the applications running on the controller are maintained, rather than blindly sending back redundant information, and reduces the quantity of data sent back to the controller during collection periods. For example, if the only application running on the controller is a routing application that calculates shortest-path point-to-point routes, the only data needed by the controller are the neighbour tables of each node. If a second application is instantiated that requires additional statistics, the controller can then reconfigure the network to include new data points during the collection period.

Signalling Reduction: A number of functions are implemented in μSDN to mitigate SDN control overhead, and are summarised in Table 3.2. As well as implementing some of the techniques and concepts proposed in works from before 2018, covered in Section 2.4, a principle

component of μSDN is its integration with the distributed routing layer. This allows it to inherently benefit from RPL Non-Storing (RPL-NS) source routing, and enables multi-hop Layer-3 forwarding without incurring routing table (or preferred parent) checks at intermediate hops. Not only is this possible when routing from the DAG root to mesh nodes but, uniquely, μSDN has the capability to directly inject source routing headers into individual packets. This key feature ensures that there is no additional SDN signalling overhead when a packet is forwarded between any two points in the mesh (unless the controller installs a contrary rule at an intermediate node), and is consistent with the minimal overhead approach observed in Figure3.1d.

Table 3.2: μSDN control signalling reduction mechanisms.

Approach	**Description**
Control Message Quenching (CMQ) [104]	Compensate for uncertainty and jitter in the control link by throttling duplicate requests to the controller (triggered by further flowtable misses) when still waiting on a response to the initial request.
Partial Packet Queries (PPQ) [105]	Ensure flowtable requests sent to the controller reference only pre-defined specific fields (rather than encapsulating the whole packet), thus preventing MTU overrun.
Source Routing (SR)	Use source routing to ensure that messages to and from a central node don't have to pass through multiple routing tables.
Source Routing Injection (SRI)	Allow the flowtable to inject SRHs directly into packets, which can then be processed either by RPL or the μSDN layer, ensuring intermediate nodes don't need to request controller instruction on how to handle the flow.
Flowtable Rule Refresh (FRR)	Allow controllers to instruct particularly active flowtable entries (i.e. X hits in Y period) to reset their lifetimers, rather than having the entry expire.

3.3.2 μSDN-UDP: Southbound SDN Protocol

μSDN uses its own lightweight protocol, *μSDN-UDP* for controller communication, utilising UDP at the transport layer to allow for secure Datagram Transport Layer Security (DTLS) when communicating with SDN controllers located outside the mesh (though the reader should be aware that all simulations and experiments discussed in this thesis utilise an embedded controller at the RPL DAG root). As discussed in Section 3.2, it is essential that any SDN control protocol for low-powered wireless networks eliminates 6LoWPAN packet fragmentation in order to minimise excess overhead. Consequently, this limits the volume of information that can be sent in each packet. To this end, μSDN uses the previously described PPQ mechanism and configurable statistics to ensure that packet size is minimised by tailoring reported information to the currently active applications on the controller.

As previously outlined, μSDN employs the same basic control processes as found in traditional SDN models: *collecting* network state information, *configuring* devices, and *reacting* to new data flows. These are mapped to the μSDN-UDP protocol, which implements them as the specific packets in Table 3.3. The periodicity of these packets are then further classified as *initial*, *synchronous* or *asynchronous* traffic, which are used to categorise traffic in the evaluation of μSDN performance in Section 3.4.

Table 3.3: μSDN packet types: showing the traffic periodicity and direction, and the category of SDN traffic.

Packet	**Periodicity**	**Direction**	**Description**
Configuration (CONF)	Initial	↓	Initialises SDN configuration settings, such as periodicity of statistics updates, default flowtable settings, etc.
Node State Update (NSU)	Synchronous	↓	Updates the controller with both local (neighbours etc.) and SDN (flowtable etc.) statistics.
Flowtable Query (FTQ)	Asynchronous	↑	Allows the node to request instructions from controller on how a particular flow should be handled.
Flowtable Set (FTS)	Asynchronous	↑	Provides a match/action pair to be installed as a rule (or set of rules) in a flowtable.

Initial (*Configuration*): μSDN employs the RPL protocol to inform the controller of nodes that have joined the DAG, and are therefore reachable. As nodes join the DAG, the controller responds with a μSDN-UDP Configuration (CONF) message. This allows nodes that have joined the network to receive initialisation information from the controller, such as: NSU timer settings, flowtable lifetimes, and set default collection statistics.

Synchronous (*Collection*): A Node State Update (NSU) message, from a node to the controller, carries information about that node, such as energy, node state, and buffer congestion. This includes observations about its immediate neighbours and link performance. These periodic messages are sent on a timer process within the *SDN Statistics* module, that can be configured by the controller through an unsolicited CONF message.

Asynchronous (*Configuration* / *Reaction*): Flowtable Query (FTQ) packets are sent from a node to the controller in response to a flowtable miss, i.e. the SDN checks the flowtable for instructions on how to handle a packet but is unable to find a matching entry. With Partial Packet Queries (PPQ), FTQ messages send a portion of the packet data up to the controller. The controller then actions that data, and transmits a response back to the sender in the form of a Flowtable Set (FTS) message. The behaviour of this traffic is by nature intermittent, though the burstiness depends on whether or not the flowtable uses Source Routing Injection (SRI). If source

routing isn't used, then it will exhibit bursty behaviour as FTQ packets are generated by each node in the path between the source and destination.

3.3.3 μSDN-Atom: Embedded SDN Controller for Fast Control Response

To prevent control signalling from having to pass through a border router onto the backbone network, and incurring additional delay on FTS responses in the case of *reaction* control traffic, a lightweight embedded controller is implemented. μSDN-Atom[3] allows simple networking applications to be implemented and run on a centrally located node in the mesh, normally as the RPL DAG root. However, the limited capabilities of an embedded Microcontroller Unit (MCU) need to be taken into account when hosting computationally heavy applications on such a constrained node. For example, a Shortest Path (SP) routing application can struggle to quickly compute paths in networks greater than ~30 nodes due to hardware restrictions. Adopting a similar approach to the μSDN stack, Figure 3.4 shows how Atom exposes abstract Southbound (SB) (connection) and Northbound (NB) (application) interfaces, and re-uses the concept of SDN *actions* in order to provide publish/subscribe messaging between these layers.

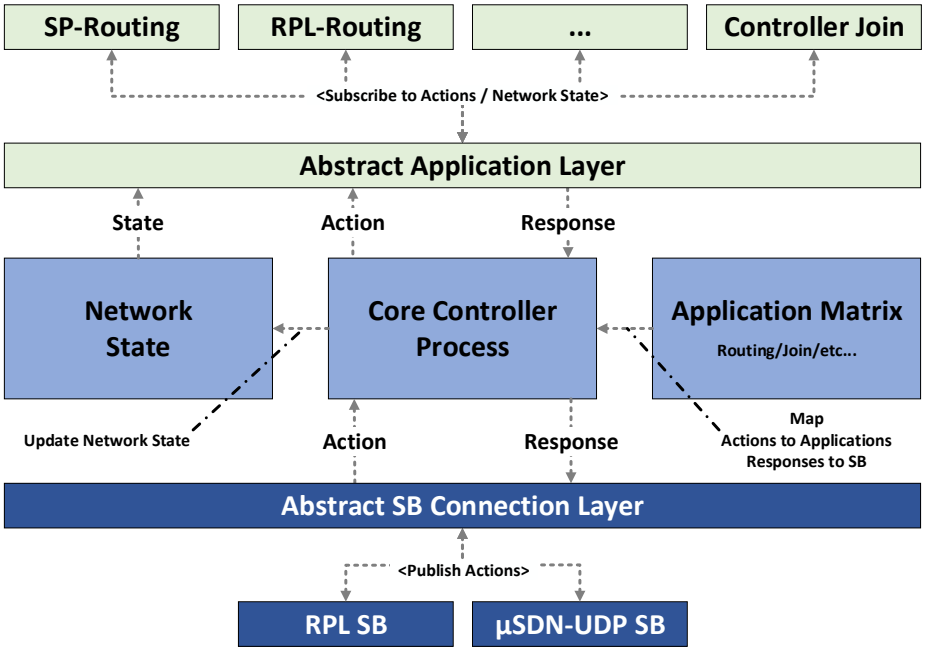


Figure 3.4: Abstract layers exposed by the μSDN-Atom embedded SDN controller. This controller is designed to run on a centralised mesh root node.

μSDN-Atom implements three default applications: the *Controller Join* applications, a *Shortest-Path (SP) Routing* application, and a *RPL Routing* application. Two SB connections

[3]N.B. this is separate work from the SF-based Atomic-SDN architecture covered in Chapter 5.

are also provided, one for receiving RPL ICMPv6 control signalling, and another for μSDN-UDP messages. There are three central Atom components: a universal *Network State* to which applications can subscribe, and that can notify the application layer of specific network changes; an *Application Matrix* that provides mapping between concrete SB connections, subscribed actions, and concrete applications; and a *Core Controller Process* that manages buffering, publishes actions to applications, and forwards responses to the appropriate SB connection.

In this manner, application logic is separated from the specific control signalling protocols. For example, following Figure 3.4 the controller association process in μSDN is initiated by the reception of a DAO at the root (which hosts the Atom controller). After being processed by the RPL (to send the DAO-ACK), the DAO is then passed to the RPL SB connection, which sends a JOIN *action* to the core process. Node information from the DAO message is encapsulated in that action and sent to the *Network State* module. The *Controller Join* application, subscribed to the *Network State*, receives notification that a new node has joined the network and starts the initial configuration process. This is pushed as a configure (CONF) *response* through the abstract application layer to the *Core Controller Process*, which checks the Application Matrix and sends the response to the SB connection that from which the JOIN *action* originated. This then sends a μSDN-UDP CONF message back to the node that sent the original DAO.

3.3.4 Summary of Architecture

Table 3.4 provides a high-level overview of all features of μSDN in comparison to the other implementations referenced in the previous chapter. This doesn't therefore mean that the table provides an in-depth overview of the features in those specific works, for which the reader should be directed to Section 2.4, but is instead to give a general summary of where μSDN fits with respect to other publicly available low-power wireless SDN architectures. Features have been grouped by benefit, in order to provide a better understanding of how the ideas and concepts that have been discussed in this chapter lend themselves to the overall μSDN architecture.

Table 3.4: Comparison of μSDN features with respect to the other SDN implementations highlighted in Section 2.4. Where information is not available, this has been denoted by '?'.

Feature	Benefit	μSDN	SDN-WISE [26]	TinySDN [112]	IT-SDN [27]	Coral-SDN [108]	Whisper [116]
Eliminate fragmentation in protocol	Overhead	✓	✓	×	×	×	✓
Reduce signalling frequency	Overhead	✓	✓	?	?	✓	✓
Message throttling (CMQ)	Overhead	✓	×	×	×	×	×
Refresh timers on active rules (FR)	Overhead	✓	×	×	×	×	×
Configurable statistics	Overhead	✓	✓	×	×	×	✓
Support for RPL	Overhead	✓	×	×	×	✓	✓
Support for IETF 6TiSCH	Overhead	✓	×	×	×	×	✓
Match on byte/index	Memory	✓	✓	✓	✓	✓	×[4]
Configurable query on byte/index (PPQ)	Memory/Overhead	✓	×	×	×	×	×[4]
Reuse matches/actions	Memory	✓	✓	?	?	?	✓[4]
Flow priority	CPU/Latency	✓	×	×	×	×	×
Hierarchical flowtables	CPU/Latency	✓	×	×	×	×	×
Full embedded SDN controller	Latency/Overhead	✓	×	×	×	×	✓
Support for distributed control	Latency/Overhead	✓	✓	✓	×	✓	✓
Source routed control	Latency/Overhead	✓	✓	×	✓	?	×
Routing layer interoperability	Resilience	✓	×	×	✓	✓	✓
Packet processing (INPP)	Configurability	✓	✓	✓	✓	✓	✓
Flowtables can inject SR headers	Configurability/Overhead	✓	×	×	×	×	×

[4]Uses policy-based control rather than match/action flowtables.

3.4 Evaluating μSDN

This section evaluates the μSDN stack on IEEE 802.15.4-2015 [1] low-power wireless networks. All simulations were presented at the 2018 IEEE Conference on Network Softwarization (NetSoft) [C1], and use an asynchronous energy-saving MAC layer, ContikiMAC [121]. This was chosen over an unslotted CSMA/CA *always-on* approach in order to evaluate the impact of μSDN on energy usage within the mesh. For experimentation and results on the IEEE 802.15.4-2015 [1] TSCH MAC option and IETF 6TiSCH[2] standard, the reader should be directed to Chapter 5.

Section 3.4.1 examines the ratio of μSDN signalling overhead generated with respect to RPL ICMPv6 control messaging, Section 3.4.2 looks at the impact of periodicity in NSU and FTS packets, and Section 3.4.3 evaluates μSDN latency, reliability, energy, and network join metrics in comparison to a SOTA RPL network. Finally Section 3.4.4 presents a use-case for μSDN through a demonstration of per-flow network slicing in an interference scenario. This evaluation not only shows that the SDN overhead can be minimised to an extent that the impact on IEEE 802.15.4 network performance is minimal, but it is further established that the programmability conferred by μSDN (and by association, SDN in general) provides distinct advantages compared to the existing low-power wireless solutions.

Table 3.5: Cooja Simulation Parameters

Parameter	**Value**
Scenario	Data Collection (all nodes)
Duration	1h
Topology	Grid
MAC Layer	ContikiMAC [121]
Radio Medium	Distance-Loss Unit Disk Graph Medium (UDGM)
Transmission Range	100m
Interference Range	0m
Receive Probability	90%
Transmit Probability	100%
Transmitting Nodes	All
Receiving Node	DAG Root/ μSDN-Atom Controller
Network Size	30 Nodes
Application Send Interval	60 - 75s
RPL Mode	Non-Storing
RPL Route Lifetime	10 min
RPL Default Route Lifetime	∞
μSDN Update Period	180s
μSDN Flowtable Lifetime	10 min

Unless otherwise stated, all simulations in this chapter were performed using the Cooja simulator for Contiki OS[5] [122], on emulated EXP5438 platforms utilising a MSP430F5438[6]

[5]https://github.com/contiki-os/contiki

[6]http://www.ti.com/product/MSP430F5438

CPU and the CC2420[7] radio, and used the configuration settings specified in Table 3.5. A variable send rate of 60-75s was used to generate dummy application layer communications such as sensor data. The network size was set to 30 nodes due to device memory limitations restricting the number of nodes at the μSDN-Atom controller, with a receive probability of 90% at each node to model channel uncertainty. μSDN update period and μSDN flowtable lifetimes, NSU and FTQ/FTS messaging respectively, were set to similar values used in default control signalling within Contiki's RPL implementation.

As these experiments focus the MAC and network layer effects of SDN control signalling overhead, a simple model was used for the underlying radio medium. Contiki's Distance-Loss UDGM model overlays a transmission disk around each node and applies a proportional receive probability (with respect to distance) to each node within range, up to a pre-defined maximum receive probability (i.e. in these simulations a node at the edge of the disk will have a receive probability of 90%). One of the benefits gained from using such a model, is that it allows physical and MAC layer phenomena to be distilled into easily quantifiable fields, while the uncertainties of real-world experimentation mean it can prove harder to examine, and compare network layer solutions - the focus of this chapter.

3.4.1 Ratio of SDN Control Overhead to RPL Overhead

Given that μSDN employs RPL for topology creation and fallback routing to the controller, it is important to consider the volume of additional control signalling incurred by μSDN in comparison to the regular ICMPv6 control overhead generated by RPL. Section 3.3.2 provided an overview of the packets used in the μSDN-UDP control protocol, sorted by periodicity. Figure 3.5 further classifies NSU collection traffic into *periodic* Constant Bit-Rate (CBR) overhead, while *asynchronous* CONF, FTQ, and FTS, messaging is grouped as Variable Bit-Rate (VBR) overhead. The ratio of this overhead, alongside the ICMPv6 control signalling generated by RPL, is evaluated with respect to application traffic in the simulation scenario described in Table 3.5.

While the μSDN-CBR (periodic) traffic clearly presents a considerable overhead within the mesh, it is considerably less than that generated by RPL ICMPv6 control signalling (consisting of DIS, DIO, and DAO messages). Furthermore, the overhead reduction mechanisms employed in μSDN manage to substantially limit μSDN-VBR (asynchronous) signalling, comprising ~3% of total traffic. This shows that although the additional control messaging from a centralised SDN architecture can't be completely eliminated, it can be reduced to levels far less than that needed by traditional topology management protocols, and represents a relatively modest overhead to the existing stack.

[7] http://www.ti.com/product/CC2420

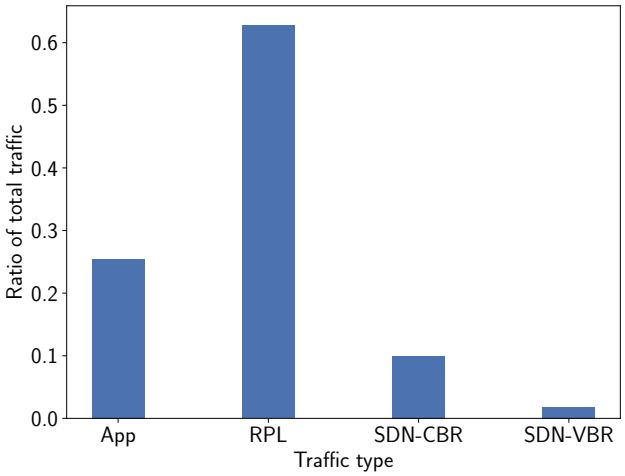
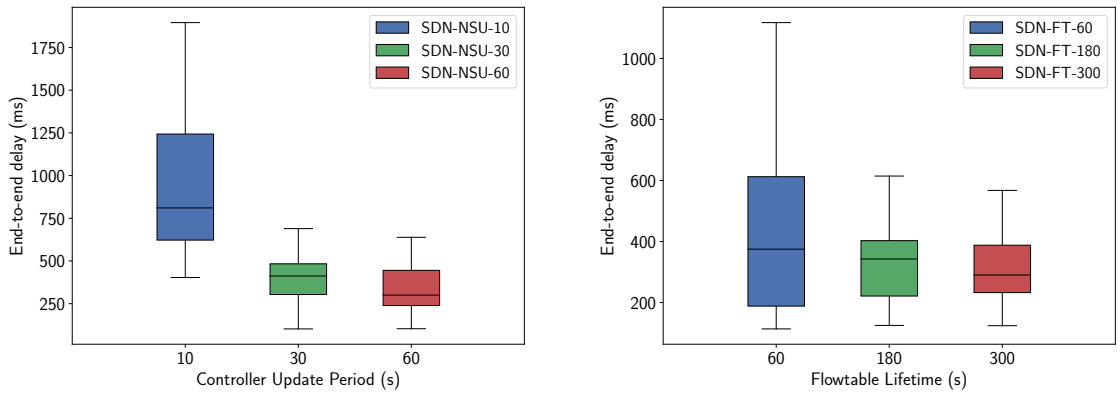


Figure 3.5: Ratio of RPL and μSDN-UDP signalling overhead with respect to traffic generated by a collection application (App). NSU messaging is categorised as Constant Bit-Rate (CBR), while CONF, FTQ, and FTS messaging is grouped as bursty Variable Bit-Rate (VBR)

3.4.2 Impact of SDN Collection and Reaction Periodicity

While Figure 3.5 presents the ratio of μSDN signalling overhead to RPL control signalling, Figure 3.6 shows network performance when not using any of the signalling reduction mechanisms introduced in the previous section, and highlights how application QoS can be sensitive to increases in μSDN traffic, firstly resulting from the periodicity of CBR NSU *collection* updates, and secondly from the regularity of VBR FTQ/FTS messages governed by flowtable entry lifetimes. In each case, the opposing parameter (NSU period and FT lifetime) was fixed at the value given in Table 3.5

While both figures show that network processes such as application traffic, as opposed to control messaging, are affected by the μSDN overhead, of particular interest are the results presented in Figure 3.6a, which show how regular state updates at 10s intervals from each node impact application QoS, while Figure 3.6b shows that 60s timers for flowtable entries have lesser but similar effects. These figures highlight that the variable and constant bit rate signalling overhead generated by the μSDN layer can have a considerable impact on other communications within the mesh. However, while the FTQ/FTS messaging generated by the expiration of flowtable entries becomes an issue as the lifetime is decreased, the NSU messaging exacts the greatest cost on network performance due to the bursty effect of all network nodes concurrently attempting to update the controller within a constrained time period, and in the process reducing resource availability for other traffic.



(a) Effect of NSU collection period on application latency. (b) Effect of flowtable lifetime on application latency.

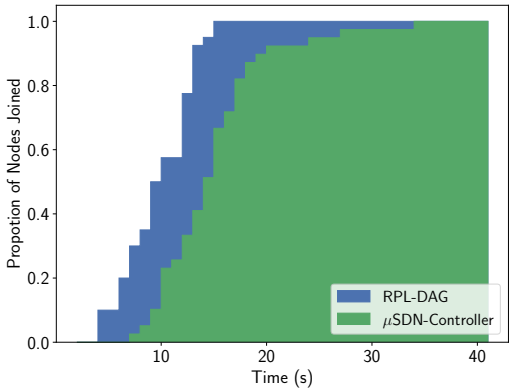
Figure 3.6: Based on the simulation parameters detailed in Table 3.5: (a) Effect of increasing NSU periodicity on application traffic delay. (b) Effect of increasing FT lifetime on application traffic delay.

3.4.3 Evaluating the Cost of μSDN Control Overhead

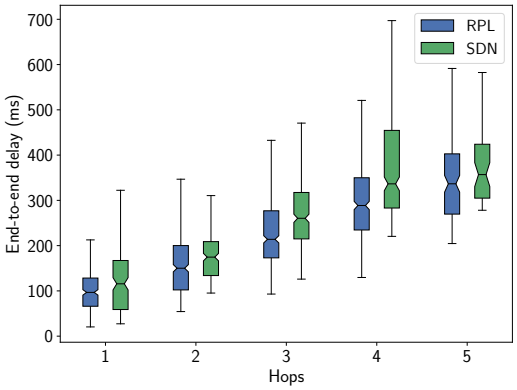
These results, as well as similar conclusions from the review of relevant literature in Chapter 2, highlight how the overhead resulting from the centralised nature of SDN control can generate considerable challenges when imposed on a multi-hop mesh network. Whether considering the availability of the radio spectrum over half-duplex links, or the memory and CPU constraints of the hardware, additional control overhead competes for limited mesh resources alongside application traffic and control signalling from other layers such as 6LoWPAN and RPL. This section therefore evaluates μSDN performance in comparison to a standard 6LoWPAN/RPL network in a typical WSN data collection scenario, with particular focus on how this overhead affects application Quality of Service (QoS), and examines performance across the following metrics:

- *Network Association*: Time for nodes to discover the RPL DAG / μSDN-Atom controller.
- *End-to-End Delay*: The effect of SDN overhead on application traffic delay.
- *Packet Delivery Ratio*: The effect of SDN overhead on network reliability.
- *Radio Duty Cycle (RDC)*: The effect of SDN overhead on node energy.

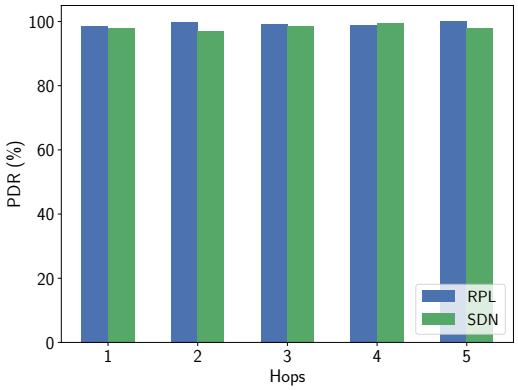
The reader should note that in this scenario μSDN has been configured with the full suite of overhead reduction mechanisms outlined in Section 3.3 (SR, SRI, FR, CMQ, and PPQ), which considerably reduces μSDN control signalling. The results of this evaluation are shown in Figure 3.7, which shows that although there is an inevitable cost in implementing SDN on top of



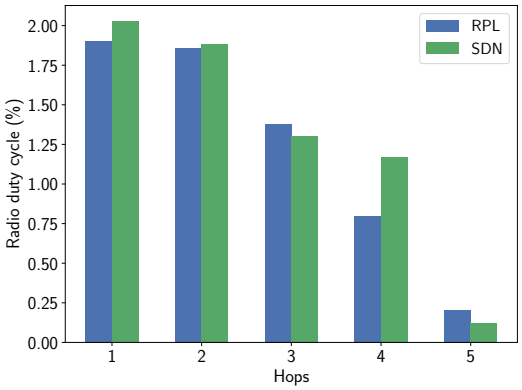
a) Join times for the RPL DAG and μSDN controller.



b) End-to-end application flow latency.



c) Packet Delivery Ratio (PDR).



d) Radio Duty Cycling (RDC).

Figure 3.7: Overall performance of μSDN in comparison to the standard RPL stack. Both evaluations were performed across a 30 node network, and follow simulation parameters outlined in Table 3.5

a distributed routing protocol in order to maintain robust connections to the controller, this cost can be minimised so that its impact on the network is limited.

Figure 3.7a presents the time taken for all nodes in the network to join both the RPL DAG, and the μSDN-Atom controller. In the case of the former, this is the time taken for the controller to learn about the routing path to that node through RPL DAO messages, which then trigger the join process.

End-to-end application latency is evaluated in Figure 3.7b. Although there is a slight increase in delay for application packets in the μSDN scenario, this is generally consistent with the slight overhead incurred by the SDN processes at each node. That is, as each node needs to perform a flowtable lookup for incoming packets, this lookup time increases the further the source node is from the destination.

Figure 3.7c shows μSDN application traffic PDR against application traffic routed through RPL. The reader should note that due to retransmission opportunities in the underlying MAC protocol, as well as the limited send rate of nodes (60-75s), the end-to-end PDR of network traffic remains high over all hop distances. Variations in PDR across hop distances occur due to random funnelling effects across the grid topology. Although such variations could be normalised in a larger network simulation, device memory limitations restrict firmware size, and consequently the maximum number of nodes supported at the μSDN-Atom controller. μSDN experiences a slightly lower PDR due to increased congestion and MAC-layer packet drops shortly after initialisation, where there is increased control signalling from both μSDN and RPL. As nodes forward application packets through SRHI they need to receive this source routing header from the controller. The increased network activity means that FTQ/FTS packets are occasionally lost, and the application packet is dropped.

Figure 3.7d shows the average duty-cycling of nodes in a 30 node network at 1 to 5 hops, where μSDN demonstrates a slight increase over the RPL case. As μSDN operates on top of the RPL protocol there is always an associated cost, particularly when considering the energy performance of nodes.

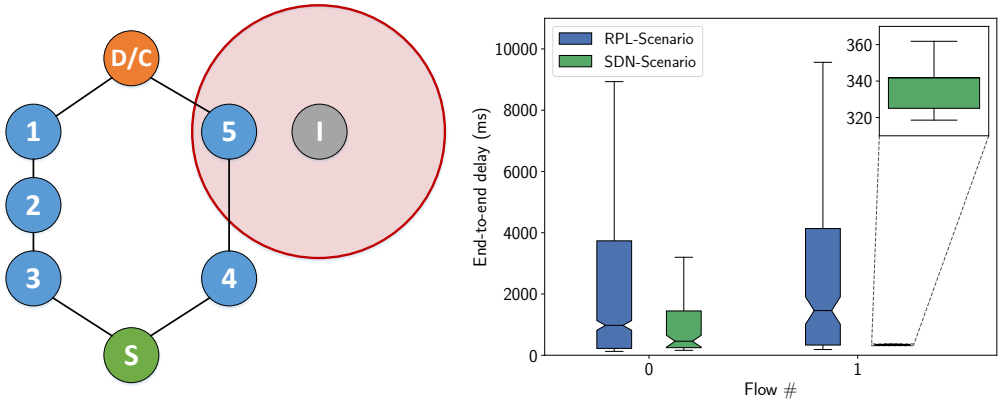
3.4.4 Demonstration of Per-Flow Network Slicing

The previous section has shown that, however minimised through the overhead reduction employed by μSDN, the additional overhead incurred by the application of SDN on the low-power wireless mesh can result in a performance hit when considering data collection scenarios typical in many IoT sensor networks.

Table 3.6: Interference Scenario Parameters

Parameter	**Setting**
Interference Period	100ms
Interference Duration	15ms
Flow F_0 Bit Rate	0.25s
Flow F_1 Bit Rate	10s

However, the configurablity conferred by SDN architecture allows for increased QoS in cases where the traditional 6LoWPAN/RPL stack will struggle. To this end, this scenario attempts to demonstrate the effectiveness of SDN's ability to programmatically control data flows and provide redundancy within the network. Table 3.6 provides the simulation parameters used in this scenario. These were chosen in order to simulate heavy wireless interference from an external device, with regular periods of contention that last considerably longer than the transmission time when using the IEEE 802.15.4 OQPSK 2.4GHz PHY option (around 4ms for a 127B MTU). Figure 3.8 demonstrates how a routing node could suffer from this heavy external interference.



a) Interference re-routing topology.

b) Interference re-routing results.

Figure 3.8: (a) Topology of intermittent interference scenario. The source node (S) is shown in green, whilst the destination/controller node (D/C) is in orange. Intermittent interference is generated at I, interfering with node 5. (b) Delay and jitter of flows in the intermittent interference re-routing scenario. Compares a μSDN scenario against a standard 6LoWPAN/RPL approach. In the former, μSDN is configured to reroute flow F_1 around the interference, and the considerable reduction in delay and jitter of critical flow F_1 can be seen in the highlighted area of the figure.

In this setup, a source node creates two flows, F_0 and F_1. F_0 is a low priority, but high volume flow, whereas F_1 is a critical flow with a much lower bit rate but high priority. RPL OF0 was used for creating the DAG topology, which instructs RPL nodes to choose their parents based on node rank (hop distance from the DAG root). In this case, the source node S will receive DAG information from both node *3* and node *4*, however it will choose *4* as its parent as that node will have a lower rank due to its proximity to the root node D, which in this scenario is both the destination node and the SDN controller. An interferer node was placed so that node *5* would experience a short burst (15ms) of interference every 100ms, causing flows across the RPL route to experience a high degree of degradation. As the interference is not constant, the RPL DAG is unable to heal and form a new path through node *3*.

The introduction of μSDN to the network allows the controller to handle flows individually, and re-route F_1 through node *3* even though it is the longer path and is not the next hop dictated by the RPL OF. Flow F_1 is therefore able to bypass the interference, experiencing reduced delay and jitter whilst Flow F_0 continues to be routed using RPL. This also has the side-effect of reducing the delay of F_0 as the path ($S \rightarrow 5 \rightarrow 4 \rightarrow D$) experiences less traffic. These results are shown in Figure 3.8b, where μSDN exhibits dramatically reduced delay in comparison to the scenario without the benefit of SDN configurability.

These results show that the configurability afforded by an SDN approach, allowing network resources (such as alternative routing paths) to be abstracted away from specific protocols and redistributed to network slices at the SDN layer, provides an effective means of handling traffic

behaviour in programmatic manner, in isolation from other data flows. This is in contrast to traditional approaches where behavioural logic is typically ingrained within the protocol itself, and it can be difficult to effectively re-provision the network for individual traffic flows.

3.5 Summary and Conclusions

The focus of this chapter has been to explore the effect of SDN control signalling on low-power mesh networks, and both propose and implement solutions to mitigating this overhead. Consequently, it has introduced the design and implementation of μSDN: a low-overhead SDN architecture that overcomes some of the major difficulties faced by SDN approaches in low-power wireless, as covered in Chapter 2. Notably, μSDN was concurrently the first SDN architecture to advocate full interoperability with both RPL and 6LoWPAN in the IEEE 802.15.4 stack, and it introduces a number of novel mechanisms to reduce the SDN signalling overhead exacerbated by the addition of this control centralisation in a multi-hop mesh.

3.5.1 Research Questions

Through this analysis, this chapter has presented material addressing research questions [Q1, Q2, Q3, Q5] and is supported by publication [C1]; while the μSDN code is made publicly available through [R1]:

[Q1]: *What are the fundamental features of a minimal SDN solution?* Section 3.2 interprets the design considerations stemming from the background material presented in Chapter 2, and distils these into four key requirement areas: *Control Plane*, *Data Plane*, *Distributed Network State*, and the *SDN Controller*.

[Q2]: *How does SDN control signalling affect the low-power wireless mesh?* The impact of SDN *collection* and *reaction* periodicity is examined in Section 3.4.2, where it is shown how frequent control signalling can have a considerable effect on delay and jitter in regular data flows. This raises the question of how SDN control signalling should be optimised in order to maximise value (such as maintaining an up-to-date view of the network state) while mitigating the cost of overloading mesh links with additional overhead.

[Q3]: *Are there solutions to reduce SDN overhead?* As well as implementing some key overhead reduction proposals from literature, μSDN is the only SDN architecture to support the injection of source routing headers from the flowtable as an approach to multi-hop Southbound communication. Through the combination of these methods, μSDN eliminates much of the signalling overhead resulting from the application of centralised SDN control across a multi-hop mesh, and Section 3.4.3 presents results showing how μSDN maintains comparable performance with respect to a RPL network in a typical data collection scenario.

[Q5]: *What advantages does SDN bring to a low-power wireless mesh?* It is demonstrated that μSDN can provide opportunities for fine-tune programmability through manipulation of SDN

flowtables. In particular, a scenario where μSDN is used to implement per-flow QoS handling within a simple network under intermittent interference is demonstrated, showing how μSDN can provide redundancy to priority flows, and considerable reduction in latency and jitter is achieved in comparison to a standard RPL approach.

3.5.2 Further Consideration

While the focus of this chapter has been to explore mechanisms to manage and reduce SDN control overhead, μSDN provides much opportunity for exploring a number of key topics within the low-power wireless SDN research area.

Controller Placement: The abstract SB connections supported by the μSDN adaptor can allow comparison between the embedded μSDN-Atom controller, and an SDN controller accessible through a border router. While μSDN-Atom delivers latency benefits by virtue of its placement within the local mesh, it is limited by the capacity of its CPU when faced with complex decision making or processing requests from a large number of nodes, and is constrained by memory restrictions in maintaining its view of the network state.

Interoperability: Similar to the approach taken in Whisper [116], there is potential for exploring SDN interoperability within a legacy mesh network in order to use SDN nodes to manipulate the traffic flowing through local branches. This could potentially reduce SDN control signalling even further, as it would only be generated by select nodes, and could allow networks to strategically place data aggregation, or, like in the use-case presented in this chapter, introduce redundancy for reliable flows. Indeed, the concept of redundant paths is one also found in 6TiSCH tracks [2], which are covered in the next chapter.

Hierarchical Controllers: Furthering interoperability considerations to incorporate the embedded Atom controller, the placement of controller nodes at common parents could allow the network to form locally controlled clusters that can make decisions without navigating a large number of hops. This could improve decision latency and reduce traffic funnelling effects further up the DAG.

Network Slicing and Function Virtualisation: μSDN nodes have the ability to filter any Layer-3 traffic through their flowtables, and while μSDN provides a couple of simple routing applications on the Atom controller, there are currently no applications that explore the full capabilities of the μSDN flowtable. As well as the standard OpenFlow actions (*forward*, *drop*, etc.) the μSDN Driver provides a rich toolkit: supporting the injection of source routing headers into flows, data manipulation through boolean operands, and invoking callback functions as a flowtable action.

Massive Mesh: Many low-power wireless SDN solutions in the current literature consider networks of less than ten nodes [27, 110, 112, 115]. Although the authors of these works don't specifically address why, the small number of nodes is likely due not only to having limited hardware devices available for experimentation, but also the overhead cost that a traditional

interpretation of SDN exacts on the overall mesh performance. While μSDN provides some element of scalability in comparison to these, it too is also constrained: both through the decision to implement a local embedded controller in order to improve latency in control decisions, as well as memory limitations at the controller limiting the number of nodes it can handle. Yet while some other works *simulate* networks of 50, 100, or even 200 nodes [26, 108, 114, 116], there has yet to be any equivalent evaluation at scale on a real-world hardware testbed. Furthermore, while such quantity of nodes is a step in the right direction, the future massive mesh networks such as those required by the industrial sponsor of this PhD are looking at networks of many millions of nodes, with local clusters numbering in the thousands. If such scenarios are to be addressed, then entirely new approaches are needed.

CHAPTER 4

ISOLATING SDN CONTROL WITH 6TISCH TRACK FORWARDING

The μSDN architecture introduced in Chapter 3 considers the impact of centralised Software Defined Networking (SDN) control signalling in an IEEE 802.15.4-2015 [1] multi-hop mesh network. It proposes and implements key mechanisms to address this overhead, and presents a use-case showing how SDN can be used to dynamically configure data flows in response to external interference. While these contributions are significant, the chapter focuses solely on *asynchronous* MAC layers such as CSMA/CA; it doesn't explore the IEEE 802.15.4e-2012 [83] amendment (later merged into IEEE 802.15.4-2015), which introduced a synchronous Time Scheduled Channel Hopping (TSCH) MAC option, or consider recent standardisation efforts from IETF 6TiSCH [2].

Chapter 2 introduced how the 6TiSCH working group has been engaged in developing scheduling processes for TSCH, which allowed the creation of channel hopping schedules but did not define how these schedules should be properly configured or maintained. By establishing scheduling mechanisms, 6TiSCH aims to provide deterministic communications [125] with minimum bounded latency for Industrial IoT (IIoT) scenarios through efficient allocation of time and frequency across the multi-hop mesh network. While both distributed and centralised scheduling are covered within the standard, the centralised approach considers concepts from the IETF Software Defined Networking [5] and Deterministic Networking (DetNet) [92] Working Groups (WGs), proposing the concept of 6TiSCH *tracks* which establish reserved Layer-2 forwarding paths across the mesh [125].

Unlike the works exploring low-power wireless SDN solutions that were outlined in Chapter 2, 6TiSCH tracks are solely focused on routing and resource allocation between two endpoints, rather than extending this concept to a centrally programmable abstraction of all mesh nodes. In contrast, low-power wireless SDN implementations such as μSDN [C1], SDN-WISE [26],

and Whisper [116] all provide a framework to dynamically configure the network in response to changing application requirements. Although 6TiSCH tracks are certainly a candidate for incorporation as part of a wider SDN architecture, on their own they fall short of offering the 'programmability' that is supported in the conventional view of SDN.

This chapter first investigates how bursty signalling from centralised control mechanisms, like SDN, can impose significant latency costs in 6TiSCH. Section 4.1 then explores how tracks, a Layer-2 slicing mechanism for creating dedicated forwarding paths across TSCH networks, could be used to isolate the SDN control paths from impacting other network traffic (and vice versa). Section 4.2 discusses how the μSDN architecture presented in Chapter 3 is ported to the 6TiSCH stack. A track reservation mechanism, as defined in the 6TiSCH standard [2], is then used to slice network resources and provide dedicated control paths across the mesh, and Section 4.3 demonstrates that although this approach can effectively mitigate the SDN control cost in a low-power wireless mesh, it also introduces new trade-offs that need to be considered.

Material presented in this chapter is supported by publication [C2]. To this author's knowledge, this was one of the first works to consider 6TiSCH centralised scheduling concepts within the wider context of low-power wireless SDN research. This research extends the μSDN framework presented in Chapter 3, implementing SDN on top of the 6TiSCH stack to explore the use of reserved Layer-2 tracks to isolate SDN control signalling from other network traffic.

Results in this chapter help address research questions [Q1, Q2, Q3, Q4], and it makes the following specific contributions:

- The chapter explores how IETF 6TiSCH embraces SDN concepts through the definition of 6TiSCH tracks.
- An algorithm is proposed to establish 6TiSCH tracks between mesh nodes and the SDN controller.
- Simulations are performed on emulated hardware, showing the impact of SDN control signalling on regular application traffic in a 6TiSCH Minimal Configuration [126] network.
- Simulations are performed on emulated hardware, demonstrating how 6TiSCH tracks can isolate signalling from a centralised SDN control architecture and mitigate the impact on other network processes.

4.1 An Overview of 6TiSCH Track Forwarding

The low-latency and deterministic properties of Layer-2 6TiSCH track forwarding offer a possible solution to one of the core challenges identified through the evaluation presented in Chapter 3: that loading the mesh with bursty signalling overhead generated by a centralised control architecture can severely degrade the performance of other network flows, such as regular application

traffic (for example, data collection in a sensor network). By establishing tracks across the paths between mesh nodes to the controller, it should be possible to isolate SDN signalling from other network traffic. Not only should this prevent SDN control from impeding on other data flows, but the deterministic nature of the TSCH schedule can then provide minimum latency guarantees for the SDN controller.

Section 2.2.3 in Chapter 2 provided a brief overview of 6TiSCH architectural concepts including schedule management, control signalling, and routing/forwarding mechanisms. The possible approaches to scheduling, forwarding and routing, are reiterated in Table 4.1, and show how the 6TiSCH proposes four mechanisms for allocating resources across a TSCH schedule: *static scheduling*, *neighbour-to-neighbour scheduling*, *remote monitoring and schedule management*, and *hop-by-hop scheduling*.

Table 4.1: 6TiSCH scheduling, routing and forwarding mechanisms from Chapter 2.

Scheduling	**Forwarding**	**Routing**
Static	IPv6 + 6LoWPAN Frag.	RPL
Neighbour to Neighbour	IPv6 + 6LoWPAN Frag.	RPL
Remote Monitoring and Schedule Management	G-MPLS Track Fwd.	PCE
Hop-By-Hop	G-MPLS Track Fwd.	Reactive P2P

Much of the work at the 6TiSCH WG has, to date, revolved around non-deterministic traffic. While 6TiSCH provides two Layer-3 forwarding options (IPv6 + 6LoWPAN) through static or distributed scheduling working alongside RPL routing, these approaches are best-effort by nature, and cannot on their own provide deterministic guarantees. The latter two options in Table 4.1 are the proposed mechanisms to establish 6TiSCH tracks, which introduce forwarding concepts defined in the IETF SDN and DetNet WGs [5, 92].

This section briefly summarises the current state of standardisation efforts concerning 6TiSCH tracks [2]. Specifically, it first outlines how TSCH resources are reserved within the schedule, then expands on the two available track allocation mechanisms: centralised *remote monitoring and schedule management*, and distributed *hop-by-hop* reservation.

4.1.1 6TiSCH Resource Terminology

As defined in Table 4.2, 6TiSCH cell resources fall under three categories: *hard cells*, *soft cells*, *shared cells*. To schedule these resources across the mesh, 6TiSCH defines the 6TiSCH Operation Sublayer (6top), which abstracts management of the TSCH schedule in a similar fashion to the Southbound (SB) abstraction layer in an SDN-based architecture [5]. To communicate with 6top, 6TiSCH defines a Layer-2 control signalling protocol, 6TiSCH Protocol (6P), that allows neighbouring nodes to negotiate the reservation of *soft cells* in order to form a Layer-2 link.

However, the 6TiSCH WG has yet to agree a protocol for SB communication with a centrally located (either on or through a root node) Path Computation Engine (PCE). Although this will likely emerge as a CoAP based protocol, it is expected to be deferred to future work, possibly as part of a IETF Reliable and Available Wireless (RAW) WG [127].

Table 4.2: 6TiSCH cell types.

Resource	**Description**
Hard Cells	Allocated by a separate control entity (such as a PCE or SDN controller). The addition, moving, and deletion of these cells can only be performed by 6top under strict instruction.
Soft Cells	Can be reserved and negotiated by two neighbouring nodes using 6P over the 6top interface. Unlike hard cells, which can only be assigned via a control entity, soft cells can be reallocated in a distributed manner.
Shared Cells	Hard or soft cells marked with a 'shared flag'. Analogous to Slotted ALOHA [93] (with the exception that the slot length is greater than the transmission time), shared cells allow nodes to compete in a slot using a back-off algorithm.

Additionally, while TSCH cells represent atomic units of radio resources, the hardware constraints of low-power devices means there is limited space available to buffer packets along a multi-hop route. The availability of buffer space can therefore restrict the scalability of 6TiSCH scheduling algorithms.

4.1.2 6TiSCH Tracks

6TiSCH tracks were initially proposed as a means of providing QoS guarantees in industrial process control, automation, and monitoring applications, and where failures or loss of communications can jeopardise safety processes, or have knock on effects on processes down-the-line. Essentially the 6TiSCH interpretation of *deterministic paths* in the IETF DetNet WG, tracks exhibit deterministic properties through the reservation of constrained resources (such as memory buffers), and the dedicated allocation of TSCH slots at each intermediate node.

Although much of the effort within the 6TiSCH WG has focused on IPv6 Layer-3 routing and scheduling, 6TiSCH tracks are a form of Generalised Multi-Protocol Label Switching (G-MPLS), where frames are switched at Layer-2 based on the ingress cell bundle at which they were received, and forwarded to a paired transmission cell bundle.

Cell bundles are groups of cells represented by a tuple consisting of *{Source MAC, Destination MAC, Track ID}*, with the number of cells within the bundle representing the allotted bandwidth for the track. Successive bundle pairs at each intermediate node create a low-latency point-to-point path between a source and destination. As packets do not need to be delivered to Layer-3, there is less process overhead at each node. In addition, the dedicated buffer and slotframe

resources means the likelihood of retransmissions and congestion loss is reduced, as frames sent along the track don't need to compete with other traffic.

4.1.3 Allocation of Track Resources

Track resources are allocated either *centrally* though a Path Computation Engine (PCE), which in the manner of an SDN controller can reserve physical mesh resources such as buffers and *hard cells*, or via a *distributed* hop-by-hop mechanism that allows nodes to asynchronously build forwarding paths through the allocation of *soft cells*. Both of the examples in Figure 4.1 show how, at each hop along the track, more than one timeslot may be scheduled for a packet (links A→B and X→Y), so as to support Layer-2 retransmissions.

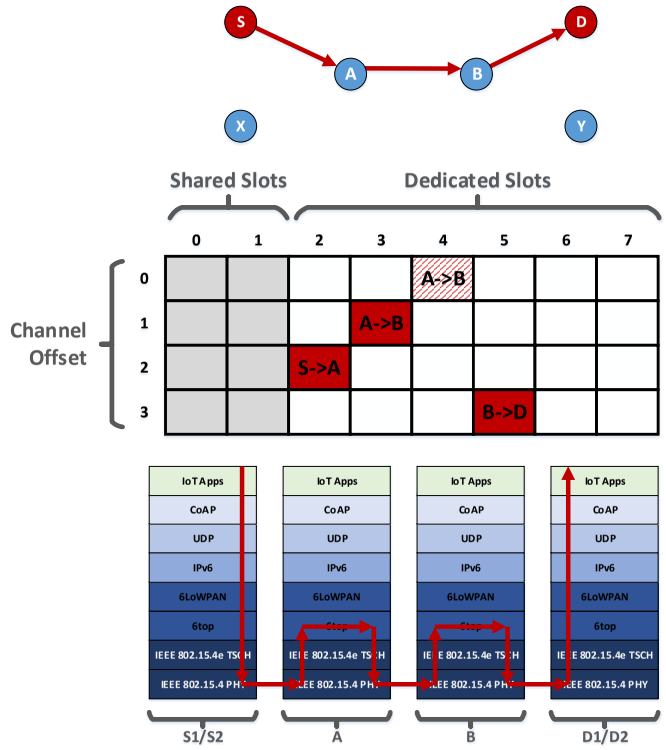
Centralised PCE Scheduling: 6TiSCH centralised scheduling relies on Scheduling Functions (SF) hosted on a PCE. The 6TiSCH Operational Sublayer (6top) interface provides a generic data module that allows the PCE to monitor and manage TSCH slot resources though CoAP. While currently the standard would consider the PCE to normally be hosted on a central entity outside the mesh, it additionally proposes that this could equally be a node multiple hops away (such as the head of a local cluster).

Distributed Hop-by-Hop Scheduling: Although centralised scheduling is capable of creating optimal forwarding paths across the mesh, 6TiSCH recognises that PCE→6top control signalling to establish these routes can create funnelling effects near the border router. Therefore, in addition to the creation of tracks through the allocation of hard cells, the standard also proposes distributed means of allocating soft cell resources through hop-by-hop scheduling across the underlying Layer-3 routing topology.

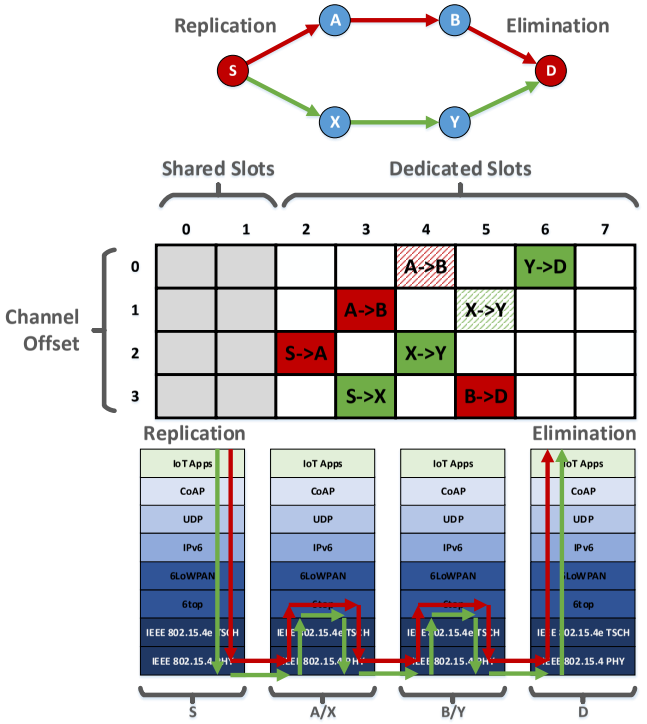
4.1.4 Serial and Complex Tracks

Figure 4.1 shows how tracks can be used to create minimal latency paths between two nodes, either through a single path *serial*, or a multi-path *complex* track. Whereas a serial track (Figure 4.1a) is a simple forwarding circuit of paired T_x/R_x bundles between two points, a complex track creates a Directed Acyclic Graph (DAG) towards a destination and encompasses the DetNet concept of Packet Replication and Elimination (PRE) [92] to establish path redundancy through spatial diversity.

Furthermore, the DetNet concept of deterministic paths extends tracks beyond the 6TiSCH mesh, allowing packet elimination not only within the mesh itself, but also on the backbone network. This idea has important connotations when considering general efforts to bring SDN to IoT. As a core principle of SDN is to abstract network operation and provide a unified control plane, a fully 'softwarized' environment (to borrow the term from Chapter 1) should allow unified control of heterogeneous networks without knowledge of the underlying MAC and physical layers. By integrating the 6TiSCH PCE within an established SDN controller architecture (like ONOS [59]



(a) Serial 6TiSCH track.



(b) Complex 6TiSCH track.

Figure 4.1: Example of *serial* and *complex* 6TiSCH track schedules. Hard cells are provisioned in dedicated slots to create a deterministic forwarding path. Additional redundant slots are allocated across links A→B/X→Y to support Layer-2 retransmissions. In (b), a complex track exploits Packet Replication and Elimination (PRE) to establish path diversity across the mesh: replicating a packet at S across the red and green tracks before eliminating redundant receptions on arrival at D

or Open Daylight [58], it should be possible to replicate tracks to multiple border routers, and then maintain guarantees until packets are eliminated elsewhere on the IPv6 backbone.

4.1.5 Interoperability with Layer-3 Routing

6TiSCH envisions tracks as a means of providing deterministic Layer-2 forwarding to highly critical flows, such as industrial actuation and alerts, where flow isolation can allow minimal latency and reliability guarantees to be provided to application processes. However, the standard also recognises the need for interoperability with Layer-3 *best-effort* routing and scheduling protocols due to the limited availability of TSCH slotframe resources. As such, if a packet received during a reserved timeslot does not belong to a track, it is then forwarded to Layer-3 for further processing where RPL and IPv6 protocols are responsible for the routing of the frame to the correct destination.

4.2 Exploiting 6TiSCH Tracks for Deterministic SDN Control

This section firstly makes a case for isolating μSDN control signalling through 6TiSCH Layer-2 track forwarding; which eliminates adverse effects on other network traffic through guaranteed and minimal latency provision of TSCH slot resources. This allows deterministic control flows to be established between mesh nodes and a centrally located SDN controller. Subsequently, this mechanism is employed to examine the application of μSDN on top of the 6TiSCH networking stack, as opposed to the asynchronous IEEE 802.15.4-2015 [1] unslotted CSMA/CA MAC layer employed in Chapter 3.

4.2.1 The Case for SDN Control Isolation

As demonstrated in Section 3.4 of Chapter 3, the overhead generated by SDN control signalling across a low-power wireless mesh can have an adverse effect on application traffic flows. Figure 3.6, in particular, showed how the burstiness of μSDN traffic (where all nodes try and communicate with the controller in a short window) can cause severe delay and jitter to other traffic flows. Table 4.3 summarises these findings, highlighting how periodic SDN control traffic generated from the *collection* services exacerbates delay, while bursty *reaction* services (in response to unknown flows) results in increased jitter as well as delay.

Table 4.3: Traffic Categorisation of μSDN Packet Types

SDN Service	**μSDN Packet**	**Behaviour**	**Impact**
collection	NSU	Periodic	Increased Delay
reaction	FTQ/FTS	Bursty	Increased Delay and Jitter

Firstly, SDN *collection* services are facilitated through Node State Update (NSU) messages, which are periodically sent from a node to the SDN controller. This *uplink* overhead carries statistical information about the state of a node, such as neighbours, energy, and link quality, and is collated at the controller to establish a snapshot on the topology and performance of the overall network. Driven by SDN control requirements, NSU messages are sent on a timer that is configured as the node joins the μSDN network. They will continue to be sent at this rate until the controller reconfigures this timer: for example, in response to new applications instantiated on the controller that require finer-grain updates.

Secondly, SDN *reaction* services are supported through a 2-way handshake process whereby Flowtable Query (FTQ) messages are sent to the controller in response to a flowtable miss (i.e. the SDN process checks the flowtable for instructions on how to handle a packet but is unable to find a matching entry), followed by a Flowtable Set (FTS) message that encapsulates one or more SDN flowtable match/action entries. As FTQ packets contain information about the packet that caused the miss, the timing of these messages is therefore asynchronous; that is, it depends on the number of 'unknown' flows seen by a node. Although this could be entirely new flows, it could also be that a node has 'forgotten' a flow due to the flowtable lifetime. When a flowtable entry expires, the node has to query the controller for new instructions about what to do with that packet type; this process is analogous to the *PacketIn*/*PacketOut* process in OpenFlow.

While this process may increase mesh overhead, it allows the SDN layer to passively react to changes in the mesh. Indeed, this is also the process used with RPL DAG Information Object (DIO) timers. However, if a number of nodes have similar expiry times for flowtable entries, this can result in bursts of queries being sent towards the controller. Moreover, although the μSDN Control Message Quenching (CMQ) mechanism presented in Chapter 3 attempts to prevent this, if the FTS response from the controller is late in arriving, then it is possible that a node could think that its original query has been unsuccessful. It then tries to send additional FTQ messages to the controller, causing possible contention through this unnecessary overhead.

4.2.2 Integration of μSDN within the 6TiSCH Stack

Figure 4.2 illustrates how the μSDN stack introduced in the previous Chapter is extended to the 6TiSCH architecture as defined by the IETF 6TiSCH WG [2]. In this figure, μSDN layers are in blue, while 6TiSCH specific layers are highlighted in green, and the standard IEEE 802.15.4 RPL/6LoWPAN stack is shaded in grey. As μSDN already sits alongside RPL, IPv6, and 6LoWPAN, there is little need for interaction between μSDN and the 6TiSCH processes since the μSDN flowtables operate at Layer-3 above 6LoWPAN.

However, in order to address the impact of SDN overhead within IEEE 802.15.4 TSCH networks, 6TiSCH tracks can be utilised to provide an isolated network slice to SDN control traffic: delivering low latency SDN controller communication with minimal jitter, and minimising disruption to the rest of the network.

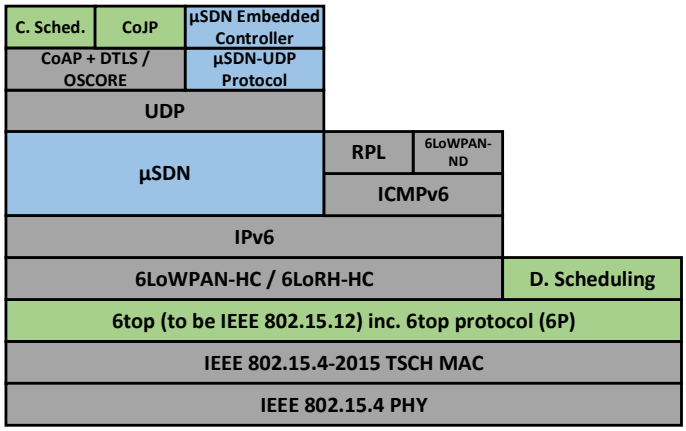


Figure 4.2: The μSDN-6TiSCH networking stack.

As discussed in Chapter 3, the unreliability inherent within the low-power mesh necessitates a distributed routing mechanism in order to provide controller discovery and maintenance, particularly when considering vulnerability to external interference. Although 6TiSCH suggests a mechanism for centrally allocating tracks through a PCE, μSDN control tracks adopt an alternate approach, and are formed using a hop-by-hop process across Layer-3 links.

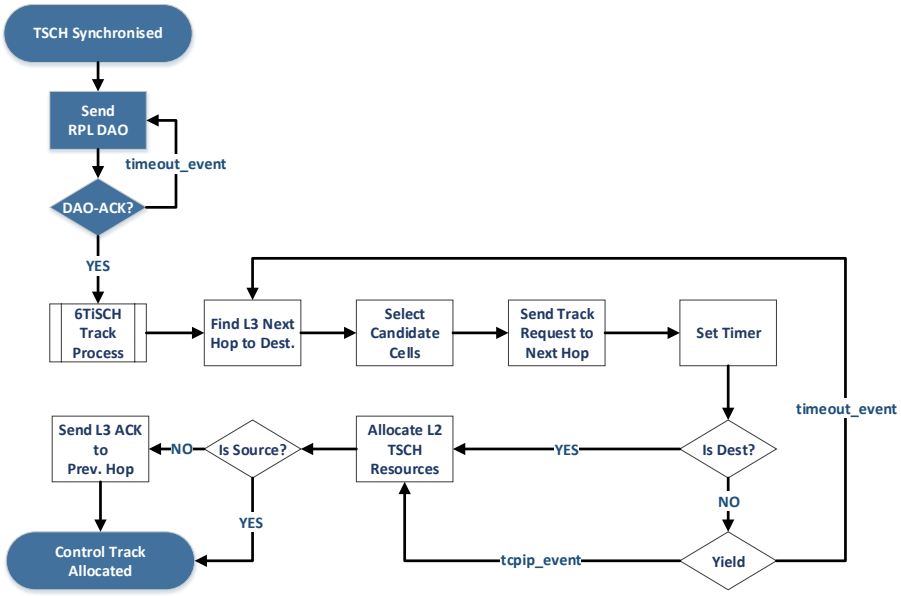


Figure 4.3: μSDN control track allocation flowchart. Control tracks are instantiated in response to the RPL DAO process, establishing *uplink* tracks to the controller once the controller has "discovered" the node through the RPL DAO.

μSDN allocates a *serial* circuit of TSCH soft cells and forms low-latency *uplink* paths along

the RPL topology towards the DAG root (where the μSDN controller is located). Figure 4.3 provides a high-level flow of this mechanism. As nodes join the controller, the 6TiSCH track allocation process is started. The node selects the next-hop neighbour towards the destination (in this case, the RPL parent); it selects candidate TSCH cells and buffer resources suggested by an allocation algorithm (in this case the same scheduling mechanism used for allocating resources for Layer-3 flows), and these resources are forwarded as candidates along to the next hop. This process is repeated until the destination is reached, or a timer expires, in which case a node reattempts (until a maximum number of times) to establish the track. When the destination is reached an acknowledgement is sent back along the circuit, and each node in the track allocates its candidate cells.

4.3 Evaluating μSDN Control Isolation in 6TiSCH

This section examines delay and jitter of SDN control traffic, as well as application data traffic, when allocating IETF 6TiSCH tracks over μSDN control paths. Results presented in this chapter have been published in [C2] at the 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).

The reader will note that these simulations consider a small network of just a few nodes. This was due to the limitations of the hardware platform required by the proprietary 6TiSCH stack: which only supports 32KB of ROM and 16KB of RAM memory. At the time of publication, these restrictions affected allocation of dedicated 6TiSCH buffers, which need to be created per-track. As an *uplink* μSDN control track is allocated for each node within the network, the maximum buffer space is quickly utilised, and limited the number of supported nodes.

Unless otherwise stated, all simulations in this chapter were performed using the Cooja simulator for Contiki OS[1] [122]. Cooja is capable of emulating the required hardware platform for the μSDN-6TiSCH stack, which runs on TI EXP5438 platforms utilising a MSP430F5438[2] CPU and the CC2420[3] radio, and requires 15ms TSCH timeslots. All parameters used for the simulations are summarised in Table 4.4. Due to the limitations on network size a linear topology was used in order to gather results across greater hop counts, in line with similar works from the research community. A variable send interval of 5-10s was used to generate dummy application layer communications such as sensor data. This is of greater rate than the previous chapter in order to compensate for the smaller network size. A receive probability of 90% was set at each node to model channel uncertainty. μSDN update period and flowtable lifetimes (NSU and FTQ/FTS messaging respectively) were set to equivalent default control signalling periods used in within Contiki's RPL implementation. All TSCH parameters used adhere to the default Contiki TSCH values.

[1] https://github.com/contiki-os/contiki
[2] http://www.ti.com/product/MSP430F5438
[3] http://www.ti.com/product/CC2420

Table 4.4: Simulation Parameters

Simulation Parameter	**Value**
Topology	Linear
# Nodes	6
Application Scenario	Data Collection
Application Send Interval	5-10s (Asynchronous)
Transmission Range	100m
Radio Medium	UDGM (Distance Loss)
Link Quality	90%
μSDN NSU *collection* traffic (Periodic)	10s
μSDN FTQ/FTS *reaction* traffic (Intermittent)	60s
RPL Route Lifetime	10min
RPL Default Route	∞
6TiSCH Scheduling	Minimal Scheduling Function (MSF) [128]
TSCH Buffer Length	4
TSCH Slot Length	15ms
TSCH Slotframe Length	31
TSCH Shared Slots	4

4.3.1 Impact of μSDN Control Signalling in 6TiSCH

Section 4.2.1 characterised how μSDN *collection* and *reaction* signalling could result in both *periodic* and *intermittent bursty* traffic when scheduled over TSCH. Due to the slotted approach of TSCH, where standard-specified timeslot is 10ms at 2.4GHz OQPSK (although Contiki can implement 15ms timeslots due to hardware restrictions), latency is bounded by the next available slotframe transmission slot. In a multi-hop network, particularly when considering funnelling effects at Layer-3 due to the RPL topology, this can result in considerable delay and jitter as SDN control packets contend with both RPL ICMPv6 messaging and application data traffic.

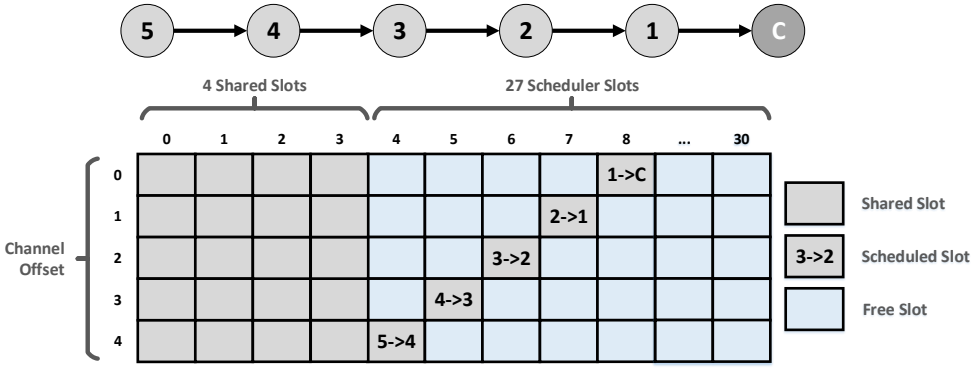
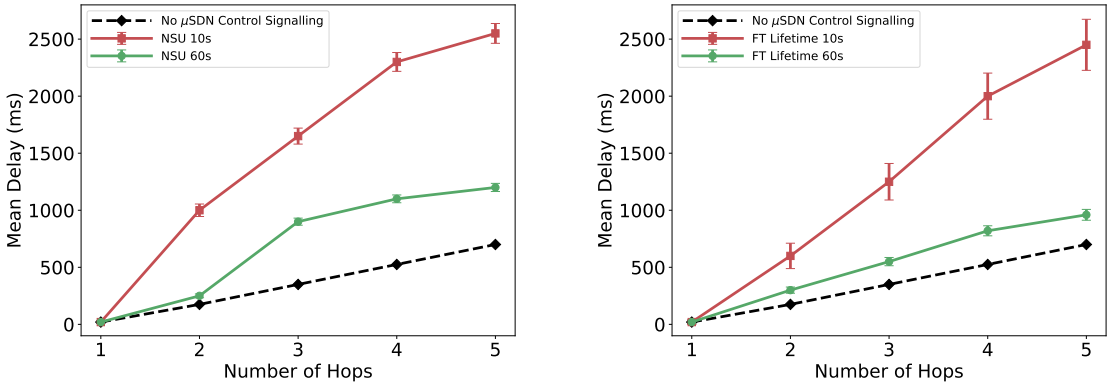


Figure 4.4: Linear topology and 32-slot slotframe used in the simulations. An *uplink* Layer-3 forwarding path is allocated using 6TiSCH MSF. The 4 shared slots are used for *downwards* traffic and TSCH Enhanced Beacons (EBs).

Figure 4.4 shows the distributed 6TiSCH scheduler estimating the network load, and creating a shared end-to-end Layer-3 forwarding path for all traffic sent *upwards* to node C, the μSDN-Atom controller. As there is no differentiation between Layer-3 traffic, all μSDN control signalling and packets from the data collection application need to share the same slots. Although the scheduler can allocate additional slots for increased traffic rates, the combination of SDN periodic *collection* services and intermittent *reaction services* can result in short periods of bursty traffic that are difficult for the scheduler to dynamically provision.



(a) Impact of NSU periodicity.

(b) Impact of FTQ/FTS periodicity.

Figure 4.5: Impact of μSDN *collection* and *reaction* periodicity on asynchronous application traffic (send interval of 5-10s) in data collection scenario.

Figure 4.5 therefore considers how the periodicity of μSDN NSU *collection* and FTQ/FTS *reaction* signalling affects regular application traffic across a 6TiSCH network in a standard data collection scenario common in wireless sensor networks. In TSCH, where transmission across a link needs to wait for the next available slot, reducing the periodicity of synchronous *collection* traffic increases the frequency of regular bursty contention periods. Figure 4.5a shows how this adds considerable delay to the data collection application traffic. Asynchronous *reaction* signalling in Figure 4.5b also introduces delay, however there is additionally a marked increase in jitter. As there are no guarantees on periodicity, nodes may send repeated FTQ messages if they don't receive a response within the μSDN Control Message Quenching (CMQ) time (as covered in Chapter 3).

4.3.2 Effect of μSDN Control Isolation

The scheduled nature of the TSCH MAC layer means that the impact of centralised SDN control signalling on the network can be considerable. However, by allocating 6TiSCH tracks towards the controller, this signalling overhead can be effectively isolated from other network processes.

Figure 4.6 demonstrates how the μSDN control track process, outlined previously in Figure 4.3, establishes dedicated *uplink* Layer-2 forwarding paths towards the SDN controller, for each node

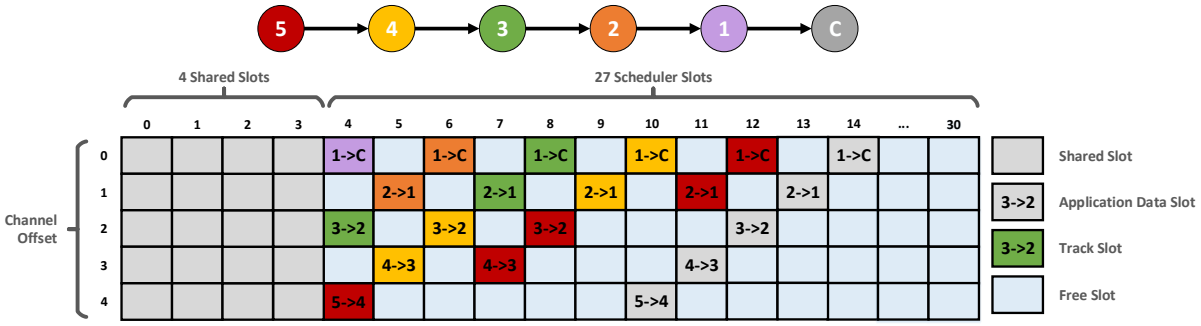
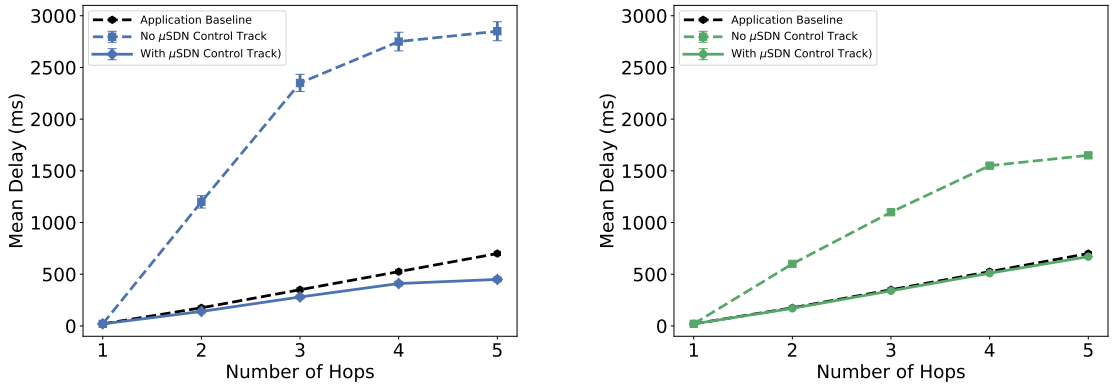


Figure 4.6: Linear topology and slotframe used in the simulations, with dedicated μSDN control tracks for each node. Each nodes' control track is denoted by colour.

in the network. Further to this, Figure 4.7 then explores the potential of tracks as Layer-2 network slices for SDN control route isolation, and their potential for reducing contention with regular network flows. Firstly, by looking at the μSDN control traffic sent over the Layer-2 tracks, and secondly at the asynchronous data-collection application traffic, forwarded at Layer-3 using slots allocated by a distributed scheduler. From the results presented in this figure, it is clear that forwarding *uplink* SDN control signalling over dedicated 6TiSCH tracks affords considerably reduced delay and jitter; both for μSDN control, as well as normal application traffic.



(a) Effect on μSDN control traffic.

(b) Effect on data collection application traffic..

Figure 4.7: Effect of SDN control overhead, showing performance of (a) SDN control and (b) application traffic in a 6TiSCH network using a distributed scheduler (optimising for minimal latency), compared to a μSDN network that provides dedicated control slices. Both results are benchmarked against application traffic in standard 6TiSCH network without SDN architecture.

As each track is essentially its own slice of the network, the SDN control overhead is isolated from the rest of the network, preventing interference and buffer contention. Interestingly, the use of tracks means that the control signalling in Figure 4.7a manages to incur slightly less delay than the benchmark (no SDN) results, particularly at larger hop distances. Firstly, buffer contention is eliminated for *uplink* SDN control messages (i.e. each node now has a deterministic

uplink path to the controller). Secondly, by forwarding at Layer-2, packets benefit from reduced processing times at each hop, as they don't incur overheads from the IPv6 and RPL routing layers. Furthermore, Figure 4.7b shows that in addition to providing a deterministic low-latency solution for isolating μSDN control, isolation through 6TiSCH tracks also mitigates its effects on the asynchronous data-collection traffic. Not only is this application traffic unimpeded by contention with the SDN control signalling, but it achieves marginally reduced delay over the baseline: due to cell reuse of the track slots when there is not μSDN control traffic.

4.4 Summary and Conclusions

This chapter has explored efforts to provide dedicated and deterministic slices for SDN control signalling over a 6TiSCH multi-hop network. It has demonstrated that by employing serial 6TiSCH tracks, dedicated Layer-2 forwarding paths across an IEEE 802.14.4-2012 [83] mesh network, it is possible to create an isolated SDN control slice for each node. This ensures that the added overhead will not interfere with the performance of other network flows, and allows SDN controller communications to benefit from deterministic networking properties as outlined by the IETF DetNet WG [92]. While the chapter doesn't provide a complete SDN solution for 6TiSCH multi-hop mesh networks, to this author's knowledge it was one of the first works to explore how aspects of the standardisation efforts from the 6TiSCH WG could be incorporated into research examining low-power wireless SDN solutions.

4.4.1 Research Questions

This chapter is supported by publication [C2], and has presented material addressing research questions [Q1, Q2, Q4].

[Q1]: *What are the fundamental features of a minimal SDN solution?* While this author would argue that 6TiSCH tracks, in and of themselves, don't constitute a complete SDN solution for low-power wireless, background research from Section 4.1 explored how the 6TiSCH concepts of *remote monitoring and schedule management*, and *hop-by-hop scheduling* have roots in the IETF SDN and DetNet WGs [5, 92]. Specifically, while wider SDN concepts of 'programmability' are not adopted (importantly, 6TiSCH centralised scheduling doesn't grant the fine-grained configurability granted by match/action *flowtables*), tracks remove reliance on Layer-3 routing protocols, and provide a loose concept for allocating TSCH resources on a per-flow basis.

[Q2]: *How does SDN control signalling affect the low-power wireless mesh?* Section 4.2 explored the issue of buffer contention between μSDN control signalling and application traffic generated through an example data collection scenario. In comparison to the asynchronous CSMA/CA MAC approach employed in Chapter 3, the availability of spectrum resources for each flow resides solely with the underlying 6TiSCH scheduling function (in this case MSF [128]). In bursty scenarios, such as with SDN control messages, packets can arrive at higher rates than the

scheduler has provisioned for. As the Layer-3 forwarding paths are shared across all traffic types, this can incur excessive delay across both control and data flows.

[Q3]: *Are there solutions to reduce SDN overhead?* This chapter introduces a mechanism through which the control plane is isolated from other network processes, by forming dedicated 6TiSCH tracks for SDN *uplink* traffic towards the controller. Although tracks don't directly *reduce* the SDN overhead, Figure 4.7 presents results showing how track isolation effectively inoculates the network from bursty SDN signalling: providing deterministic control paths; and eliminating slot contention between RPL ICMPv6 messaging, application data, and μSDN control services.

[Q4]: *Can SDN scale in a low-power wireless mesh?* While the results presented in this chapter show that 6TiSCH tracks can effectively slice the network and provide SDN control isolation, the simulation only considered a limited number of nodes, due to the memory constraints of the underlying hardware platform. When observing the tracks allocated in Figure 4.6, it can be clearly identified that as the network grows the process of allocating a dedicated track for each node is unsustainable within a given slotframe size, particularly at larger hop distances. While scalability is undoubtedly a fundamental limitation with this approach, there are a number of possible solutions. For example, inspiration could be taken from 5G, where scalability around slicing has already been addressed in literature through techniques such as clustering and shared slices.

4.4.2 Further Consideration

While this initial work on SDN control isolation within a low-power wireless mesh considered 6TiSCH tracks as a solution for mitigating the overhead of bursty control signalling, there are both some fundamental issues with this approach (as alluded when discussing research question [Q4]), and additional considerations that have not been explored in this chapter.

Extending Tracks Across the IPv6 Backbone: As discussed in Section 4.1, although not fully defined within the 6TiSCH standard, the Deterministic Networking (DetNet) [92] model specifies that it should be possible to extend tracks beyond the mesh network. When considering the conclusions made in Chapter 3, which identifies that some SDN control tasks might necessitate a control entity hosted outside the network, this could be particularly advantageous for time-critical tasks like SDN *reaction* services.

Scalability: As highlighted in research question [Q4], allocating a dedicated 6TiSCH track for each SDN control path is unsustainable in larger networks. However, this chapter has only considered the use of *serial* tracks for *uplink* SDN control signalling. Not only can tracks forward traffic to multiple endpoints, which would accommodate *downlink* SDN traffic (e.g. μSDN CONF and FTS packets), but extending 6TiSCH Packet Replication and Elimination (PRE) concepts to allow multiple source nodes to share the same *complex* track could help alleviate the considerable slot resources required in the approach outlined in this chapter.

Further Standardisation Activities: While this initial research was performed in 2017, tracks

are yet to be fully defined within the 6TiSCH standard. Given that the group's activities are drawing to a close, it has been suggested that parts of the 6TiSCH track concept be moved to a new Reliable and Available Wireless (RAW) WG [127], while 6TiSCH itself could be rechartered to specify how exactly tracks are allocated within the TSCH slotframe.

CHAPTER 5

ATOMIC-SDN: A HIGH-RELIABILITY, LOW-LATENCY SDN CONTROL PLANE

Chapters 3 and 4 introduced μSDN, covering its design and implementation for IEEE 802.15.4-2015 [1] CSMA/CA networks (as implemented by typical modern WiSUN [74] deployments), as well as exploring the potential of scheduled MAC approaches like 6TiSCH [2] to provide a means of establishing guaranteed control paths between the SDN controller and mesh nodes. Yet both chapters also demonstrate how difficult it is to apply a centralised SDN architecture, which requires frequent communications to and from a central control entity, to low-power wireless mesh standards such as IEEE 802.15.4. Even with a lightweight architecture such as μSDN, which co-opts distributed routing protocols to ensure control link integrity and employs a number of mechanisms to reduce unnecessary or excessive control messaging, the shared nature of the underlying wireless medium, a need to transmit data over multiple hops, and stringent resource constraints pose significant challenges not present in the conventional wired SDN networks. That unavoidable feature of SDN, a centralised control architecture, monopolises already scarce mesh resources and inevitably results in trade-offs and contention with other protocols.

However, as discussed in the initial chapters of this thesis, if these challenges can be addressed, and the configurability that SDN brings to traditional wired networks can be applied to the complex environment of extremely large IoT mesh networks, then it can provide a path for wireless mesh to move away from a single-application, single-tenant model to something far more dynamic: provisioning services guarantees for multiple applications and multiple multiple tenants on a single infrastructure.

This chapter introduces a highly novel approach to solving these challenges. Specifically, Section 5.1 briefly revisits the concepts of Concurrent Transmissions (CT) and Synchronous

Flooding (SF) covered more extensively in Chapter 2. Although SF in and of itself is not novel, the success of SF for robust and low-latency communications suggests that its application in solving the problem of SDN control for low-power wireless networks is highly promising. Through examination of how SF protocols can be used to propagate messages across the mesh with extremely high-reliability and low-latency, Section 5.2 then makes the case for utilising SF as the basis for SDN control in IEEE 802.15.4 low-power wireless networks. Consequently, the design and implementation of Atomic-SDN is presented in Section 5.3: a low-latency SDN architecture based on the SF concepts introduced in Chapter 2. In this manner, Atomic-SDN provides low-power mesh networks with an SDN control plane capable of broadcasting to all nodes with extremely high-reliability and at theoretical lower bounds of latency. Finally, the scalability and performance of Atomic-SDN in comparison to other approaches is demonstrated in Section 5.4 through simulation and testbed experimentation. It is shown that, in comparison to other SDN implementations for low-power wireless, Atomic-SDN maintains extremely high-reliability and minimal latency as the local mesh scales.

This work draws on the author's experience in implementing SF solutions for the International Conference on Embedded Wireless Systems and Networks (EWSN) Dependability Competition [DC1, DC2], where a version of Atomic-SDN placed 2^{nd} for both *data collection* and *data dissemination* categories in 2019, and was the only entry to achieve 100% reliability in extreme jamming scenarios. Subsequently, this chapter presents material that addresses research questions [Q3, Q4], and makes the following specific contributions as supported by publications [J1, C3] and patent application [P1]:

- SF is proposed as a novel approach for facilitating SDN in low-power wireless networks; providing one-to-all broadcast communication patterns, extremely high-reliability, and low-latency for SDN control signalling.
- A flexible SF middleware system is devised: allowing the design, instantiation, and scheduling of multiple SF protocols. This solution is used to facilitate the complex control requirements of SDN in low-power wireless, where no single SF protocol is capable of satisfying the plurality of traffic patterns.
- Atomic-SDN is presented: a scalable SDN architecture that offers considerable improvements in reliability, latency, and energy efficiency over current low-power wireless SDN architectures.
- Atomic-SDN is implemented in Contiki for the TI MSP430F1611 Microcontroller and CC2420 radio.
- Atomic-SDN is evaluated through simulation against μSDN and SDN-WISE, where it's shown that SF can significantly improve the utilisation of network resources with significant performance gains in comparison to the use of conventional MAC approaches.

- Atomic-SDN is evaluated on a 19 node real-world testbed, where it's shown that it can provide reliable SDN control under interference.

Atomic-SDN represents considerable coding effort and a mature communications stack that has been extensively 'battle-tested' in the Dependability Competition as well as subsequent publications. While Atomic-SDN has not been publicly released (as with μSDN), it is an active and ongoing research effort with the sponsor of this PhD and is currently employed as the base communications framework across five different projects: from supporting mesh communications across an IoT testbed, to providing synchronisation services for multi-hop scheduling protocols. Indeed, the most promising aspect of this approach (Synchronous Flooding) is that it can be used as a temporally decoupled distribution mechanism for control and scheduling information to an IoT mesh (covered in Section 5.2.3). When paired with a TDMA schedule, such as IEEE 802.15.4-2015 TSCH, this could be used to fully realise multi-tenant mesh solutions, and is suggested as a promising research avenue in Section 6. Since the completion of this PhD, Atomic has been extended to provide a universal mesh network scheduler capable of switching MAC and PHY solutions on single radio device.

5.1 Revisiting Synchronous Flooding: A Brief Overview

Atomic-SDN leverages recent research on Concurrent Transmissions (CT) and Synchronous Flooding (SF) to provide a high-performance platform for SDN control in low-power wireless networks. SF protocols are key to the success of Atomic-SDN, allowing the SDN controller to concurrently configure the entire network with high-reliability and minimal latency. Although background on CT and SF is covered in detail in Chapter 2, this section revisits some of the salient points, in order to support the arguments that follow.

5.1.1 What are Concurrent Transmissions?

Concurrent Transmissions describe the process whereby two or more nodes simultaneously attempt to transmit data to a receiving node at precisely the same time and on the same frequency. In forgiving low-power wireless physical layers, such as the IEEE 802.15.4 2.4GHz OQPSK-DSSS PHY layer highlighted in Chapter 2, if these concurrent signals are sufficiently time-synchronised (to within 0.5μs when using 256kbps OQPSK-DSSS) [6]) and are sending the same data, then they can be reliably demodulated despite phase offsets introduced by the competing transmissions. Additionally, this physical layer option experiences significant capture effects [65] through the 8-to-1 chip to bit redundancy in DSSS [1]. This allows concurrent transmissions of different data to have a chance of being demodulated at the receiver as long as one of the signals is around 3dB greater than the others. However, the reader should be aware that this property is not experienced as significantly in other physical layers such as those used in Bluetooth 5 [78]. This therefore raises challenges in how CT is employed across various low-power wireless

communications technologies, as CT protocols that also rely on the capture effect cannot be immediately applied to other physical layers.

5.1.2 What is Synchronous Flooding?

Synchronous Flooding describes the area of research focused around the application of CT to provide highly reliable and low-latency flooding protocols for low-power wireless mesh networks. From the initial publication of Glossy [6], where the authors showed that CT could be used to initiate a time-synchronous flood of a multi-hop IEEE 802.15.4 low-power mesh, there have been numerous SF protocols proposed and implemented that together cover the plurality of traffic patterns in a multi-hop mesh (*one-to-all*, *one-to-many*, *many-to-one*, *one-to-one*) [94]. The highly synchronised nature of SF protocols, and their ability to ignore contention issues that normally complicate low-power mesh communications, allows them to rapidly propagate a packet across the mesh with minimal latency. Furthermore, the inherent broadcast nature of flooding, as well recent advances in applying slot-based channel hopping within a single flood [7, 8, DC2], means SF protocols enjoy extremely high-reliability through a trifecta of aggressive temporal, spatial, and frequency diversity.

5.2 The Case for a Synchronous Flooding SDN Control Plane

Software Defined Networking (SDN) allows network services to be centrally programmed onto functionally agnostic hardware. The ability to reconfigure the network as needed, quickly install new protocols, or slice network resources across applications and tenants, allows networks to adapt to changing requirements or shifts in business needs. As discussed and identified in Chapter 2, to fully replicate these capabilities on a low-power wireless mesh, an SDN architecture should be able to provide the three core *collection*, *configuration*, and *reaction* services.

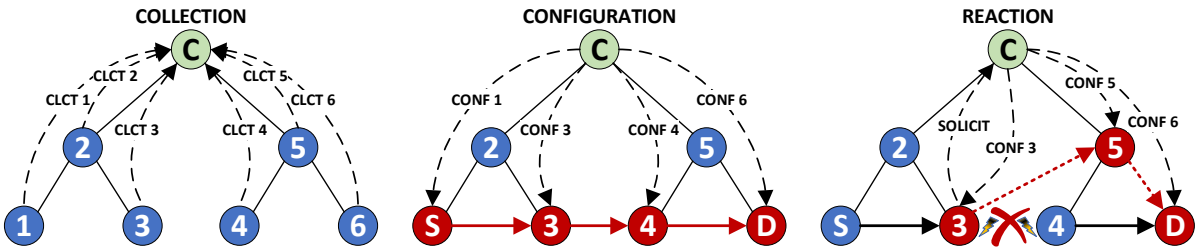


Figure 5.1: Core SDN services as identified in Chapter 2: *Collection* (CLCT), *Configuration* (CONF), and *Reaction* (SOLICIT + CONF). Nodes that need to receive instruction from the SDN controller are marked in red. Nodes *S* and *D* mark the source and destination nodes for a point-to-point link across the mesh.

This requires the network to support multiple traffic patterns (*one-to-many*, *many-to-one*, etc.), where each pattern consists of either *uplink* or *downlink* communication from nodes to

the controller, or a combination of both. In the first case, *uplink* communication allows the controller to gather network state data from devices. In the second, *downlink* communication allows the controller to configure nodes within the network. By using a mixture of these, core SDN operations can be executed. However, as alluded to in the introduction of this chapter, the centralised nature of SDN means that applying it within a multi-hop mesh complicates the process of implementing these patterns, which are not easily supported at Layer-2/3.

A radically new approach is required; one that minimises the impact of control signalling overhead while supporting all traffic patterns. This new approach ultimately needs to remove the complexities of mapping traditional SDN architecture to the currently available protocols in low-power wireless; where fundamental challenges arise from the controller not only having to communicate reliably with all nodes, but that each individual operation (for example, to set a path between two nodes) can mean the propagation of control messages across multiple hops in order to correctly configure the network. Synchronous Flooding (SF) can meet these challenges. Over the past few years SF has been shown to be extremely capable in delivering fast, reliable communications in low-power wireless networks, and the potential of SF protocols has been demonstrated through their dominance of a yearly Dependability Competition held as part of IEEE Embedded Wireless Systems and Networks (EWSN). Evaluating wireless protocols in terms of *reliability*, *latency*, and *energy-efficiency*, the competition invites teams of researchers to submit their low-power wireless solutions to rigorous benchmarking on a ~50 node real-world testbed, where they are then subjected to increasing levels of interference across the mesh. Since its inception in 2016, SF solutions have consistently beaten other approaches in terms of both reliability and latency [7, 8, 129–131, DC2], and this success supports the case for exploring the use of SF as a platform for SDN control in low-power wireless.

This section outlines how SF solutions can help to meet the challenges of delivering the SDN *collection*, *configuration*, and *reaction* services, highlighted in Figure 5.1, in a low-power multi-hop mesh. These services require frequent back-and-forth communication between the SDN controller(s) and network nodes. The flow of this traffic follows a variety of different patterns, including *many-to-one*, *one-to-many* and *one-to-one* communication, as well as exhibiting both periodic and bursty characteristics. Unfortunately, neither conventional Layer-2 approaches, nor use of standard Layer-3 topology protocols such as RPL (as employed by μSDN in Chapter 3), provide optimal performance for the plurality of all SDN traffic.

5.2.1 Topology Agnostic

Currently, multi-hop mesh networks often rely on underlying protocols such as RPL to construct a network topology. This distributed routing protocol builds a tree-like Direction Orientated Directed Acyclic Graph (DODAG), which is typically used to funnel *many-to-one* data *upwards* from the sensor network towards a single border router where they are later processed.

However, *one-to-many downwards* communication is a common issue in RPL networks. An

Table 5.1: Summary of Advantages Enjoyed by SF Protocols.

Advantage	**Description**
Topology Agnostic	As flooding allows *one-to-all* broadcast, control signalling is reduced, mobility is supported, and there is no requirement for communications scheduling.
Minimal Latency	All communications are shortest-path by nature, and Back-to-Back (B2B) transmissions can allow a packet to be sent with theoretical minimum bounds on latency.
High-Reliability	Slot-by-Slot channel hopping, combined with minimal slot times, means an SF flood enjoys aggressive temporal, spatial, and frequency diversity.
Temporal Decoupling	The time-synchronised approach of SF allows guarantees to be made on the length of a flood, allowing floods to be scheduled alongside other network operations. This could, for example, allow a SF flood to be scheduled and encapsulated within a TSCH slot.

example of this challenge is shown in the *configuration* scenario in Figure 5.1. In this case, the SDN controller wishes to set a Point-to-Point (P2P) link from S→D across multiple branches of the RPL DAG. The tree-like topology forces the controller to navigate multiple branches to configure all destinations, resulting in packet duplication as it individually transmits to each child. This is particularly relevant in RPL-NS (Non-Storing) mode which doesn't support multicast forwarding, although recent efforts attempt to address this [132]. This issue isn't specific to SDN in low-power wireless, however the complex requirements of SDN control means it is a highly visible and present issue for SDN implementations based on IEEE 802.15.4 networks.

Conversely the *one-to-all* broadcast nature of SF means it can reach all network nodes in a single flood. Unlike conventional approaches, this renders SF protocols inherently stateless, without the need for routing over any particular network topology. Rather than having to send multiple messages across multiple branches in order to reach all destination nodes, SF protocols can reliably configure all participating nodes within a single message. By virtue of this agnosticism, SF protocols forgo reliance on regular topology management control signalling, inherently allow for mobility, and provide resilience against link uncertainty.

5.2.2 Minimal Latency and High-Reliability

Chapter 2 explored how recent works have established that, through aggressive temporal, spatial, and frequency diversity, SF protocols are able to propagate a packet across the mesh at the theoretical minimal bounds on latency, with extremely high-reliability.

Minimal Latency: As there is no need to route across a network topology, all communications are inherently shortest-path, with minimal latency bounded by the number of slots in a single flood. Moreover, the use of CT for underlying communications means SF protocols don't suffer self-interference from neighbouring hidden, or exposed nodes as shown in Figure 5.2, and unlike

TSCH based approaches avoid the complex problem of scheduling co-located communications.

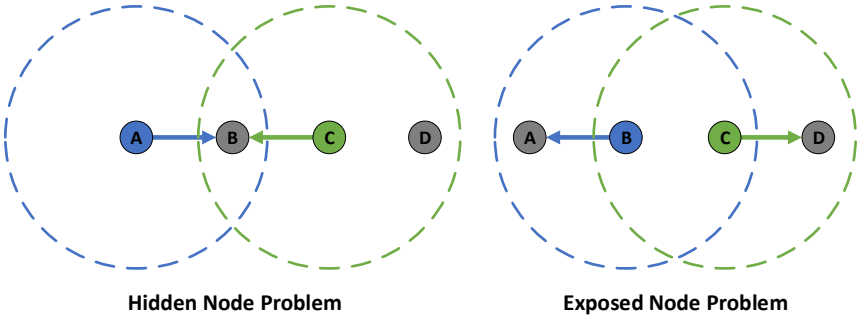


Figure 5.2: The hidden/exposed node problem can cause contention issues in high data-rate or dense IoT mesh deployments that rely on asynchronous MAC layers such as CSMA/CA.

In the hidden node problem, A and C both wish to transmit to B. As they are out of range of each other, their Clear Channel Assessment (CCA) check will allow them to transmit, but the two transmissions will interfere when received at B. In the exposed node problem, B and C both wish to transmit to separate destinations (A and D) at similar times. However, when one node transmits first (B, for example), this causes the CCA check to block the other node (C) from transmitting; if there is a high data rate, this can cause delays at that second node. Though these two issues can present significant challenges for other MAC approaches, CT inherently relies on nodes transmitting at exactly the same time, and is not affected by either of these problems. By using Back-to-Back (B2B) transmissions as shown by Figure 2.23 in Chapter 2, SF protocols are therefore able to immediately relay a received packet across the multi-hop mesh in the shortest time possible.

High-Reliability: Aggressive channel hopping techniques mean SF protocols are extremely robust against both fading and sources of external interference, common in low-power narrowband communications such as IEEE 802.15.4-2015 [1]. While synchronised TDMA approaches such as WirelessHART [72] and 6TiSCH [2], and some asynchronous MAC layers [133, 134], can provide frequency diversity through channel hopping [135], the tightly time-synchronised nature of SF means that a slot-by-slot channel hopping scheme (explored later in Section 5.3.5) allows SF protocols to more easily escape interference or poor channels. Furthermore, the inherent broadcast nature of flooding confers a great deal of spatial diversity to communications. In many scenarios, this allows messages to skirt around interference hot-spots without retransmitting, thus saving on energy and decreasing latency. This combination of frequency and spatial diversity is particularly relevant when operating over the shared 2.4GHz band which, in close proximity to external high-power devices such as IEEE 802.11 [123], can pose significant interference to low-power wireless networks.

5.2.3 Temporal Decoupling

Due to its time-synchronised nature, an SF-based control plane allows SDN services to be decoupled from other network processes. In current MAC approaches, as neighbouring nodes within a low-power wireless network share a single link over half-duplex radios, any additional control messaging increases contention over scarce resources, causing delay and reduced reliability for other control protocols (such as RPL and 6LoWPAN) as well as application data. As covered in Chapter 2, recent works have tried to mitigate the burden of this overhead through various means such as reduction in the number of control messages [26, 27], the use of source routing headers and dedicated forwarding paths [C1, C2], optimisation of the mesh routing protocol [108], and the strategic placement of SDN nodes [116]. However, all these approaches (including the μSDN architecture from Chapters 3 and 4) limit the effectiveness of SDN architecture: sacrificing responsiveness and fine-grain configurability for performance and scalability. SF provides a highly reliable means of simultaneously communicating to all network nodes, and completely isolating this overhead to free-up network resources.

Figure 5.3: Atomic-SDN uses time-sliced SF control to maximise network resource utilisation during control periods, and free-up resources for other network processes. SDN *collection* (CLCT), *configuration* (CFG), and *reaction* (REACT) opportunities are preceded by an *indication* (IND) flood that informs the network of the type of SDN control service that will follow.

As previously shown in Figures 2.21 and 2.23 in Chapter 2, the start and stop time of each flooding round is known globally across all nodes in the mesh. This allows an SF network to temporally decouple the flooding process from normal network operation, ensuring that SDN control messages are not in contention with other processes. Moreover, these messages can be bounded in time within a single flood, or distributed across multiple flooding rounds. Figure 5.3 shows how this is achieved in Atomic-SDN, which can schedule SDN services in-between normal network operation. The low-latency, high-reliability broadcast messaging that SF provides means the SDN controller can dependably configure the mesh within a single flood, and provides guarantees on how long a SDN control operation will take, as well as the maximum energy that will be expended in execution of that operation.

When paired with frequency and time division MAC layers such as IEEE 802.15.4 TSCH, the ability to temporally decouple control operations and bound them within a known time provides an opportunity to research how SF-based control platforms (such as Atomic-SDN) can potentially provide multi-tenant solutions for IoT mesh networks. While research activities from

6TiSCH have developed a framework for multi-tenant and multi-application support in IoT mesh, centralised scheduling has remained relatively untouched. Although this is not explicitly explored within this thesis, it is suggested as an avenue for further research in Chapter 6, and is a key interest of this thesis' industrial sponsor.

5.2.4 An Alternative to Other MAC Approaches

Current approaches to SDN in low-power wireless have mostly been built on the IEEE 802.15.4 network stack using the 2.4GHz OQPSK-DSSS physical layers. Although some solutions are implemented on top of the *synchronous* Time Scheduled Channel Hopping (TSCH) MAC layer [115, 116, C2], the majority employ *asynchronous* approaches such as unslotted CSMA/CA [26, 105, 108, C1]. In either case, this choice has direct impact how the SDN control plane communication is supported in terms of delay and reliability.

Asynchronous MAC Approach: The use of an asynchronous MAC layer allows the SDN control architecture to flexibly support non-deterministic traffic and is therefore suited to providing low-latency SDN *reaction* services: for example, when nodes need to be able to solicit the controller in order to request instruction on how to deal with unknown flows. However, this approach is best-effort by nature, with multiple protocols (both data and control) competing to transmit in an opportunistic manner. This consequently introduces contention within the network. Different MAC solutions have various means of dealing with this issue, typically through application-layer acknowledgements and retransmissions, yet this can still result in end-to-end packet losses of up to several percent. Additionally, these retransmissions increase the overall energy consumption of the network, as nodes must stay awake for longer. The fundamental issue therefore becomes one of trade-offs between performance and efficiency, which are subsequently inherited by SDN architectures based on this MAC approach.

Synchronous MAC Approach: Through a time-synchronised MAC like IEEE 802.15.4e-2012 TSCH [83], traffic can be scheduled in both time and frequency. In terms of SDN control, this synchronous approach suits the deterministic traffic behaviour exhibited in the data *collection* processes necessary for the SDN controller to keep an up-to-date network view. TSCH based SDN has recently been explored in Whisper [116], as well as in Chapter 4 of this thesis, where the synchronised TSCH schedule is used to slice radio resources and create contention-free SDN control paths across the mesh. However, TSCH based networks are less suited to managing the bursty asynchronous traffic generated by SDN *reaction* services. Specifically, TSCH can either provide dedicated slots for each individual link on a TDMA basis, or nodes can contend during shared slots like in Slotted ALOHA. In either case, control traffic must wait until an allocated slot in order to transmit, incurring a minimum delay overhead on the link and reducing controller response times.

SF MAC Approach: Atomic-SDN overcomes the challenges SDN faces when using either of the approaches above. Specifically, the stateless network communication made possible by

SF satisfies two issues. Firstly, although it is scheduled like TSCH, disregarding contention issues allows Atomic-SDN to reduce the slot size and rapidly propagate controller information across multiple hops within a single flood, reducing the total network resources required in an *upwards* data collection scenario. Secondly, the broadcast nature of the flood allows Atomic-SDN to eliminate contention in the *downward* data dissemination scenario, as the controller can simultaneously transmit to multiple nodes. Finally, the high-reliability of SF protocols minimises retransmissions, further reducing the control overhead and freeing schedule resources for other network processes.

5.3 Atomic-SDN Design

While the clear benefits of Synchronous Flooding (SF) protocols (namely *minimal latency*, *high-reliability*, and *zero reliance on network topology*) provide motivation for an SF-based approach to SDN control, the body of SF protocols that have been proposed to date have been highly-tailored to specific traffic patterns and application Quality of Service (QoS) requirements. As SDN requires multiple different traffic patterns in order to satisfy the various SDN services, this poses the question of how multiple SF protocols can coexist within a single architecture. In response to this challenge this author developed a novel SF architecture, supported through patent application [P1], that is able to dynamically construct, configure, and schedule different SF protocols in response to changing SDN control requirements.

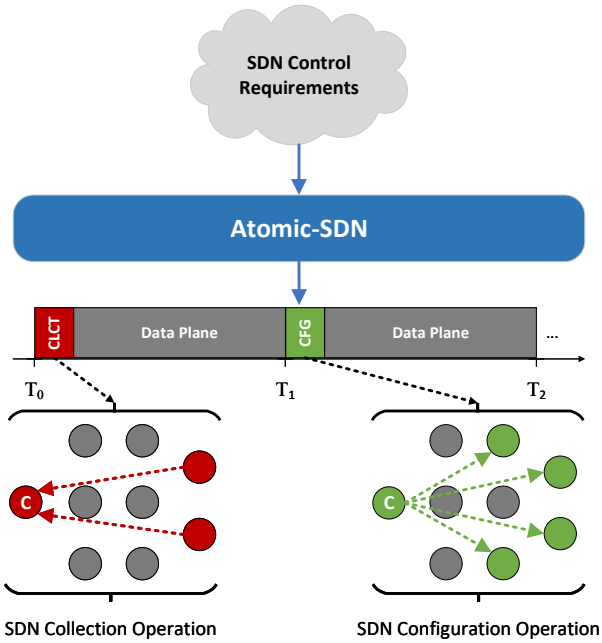


Figure 5.4: High level overview of the Atomic-SDN approach. SF control slices allow for minimal SDN control overhead, whilst a novel architecture means SF protocols can meet requirements for the plurality of SDN control operations.

Briefly summarised in Figure 5.4, Atomic-SDN separates SF protocol logic from the lower-layer radio operation, by defining an atomic flooding primitive on which more complex protocol operation can be built. Applying pre and post logic to this primitive, a basic building block is defined from which higher-layer protocol logic can be constructed by chaining various blocks together. In short, this allows multiple different SF protocols to be dynamically defined and scheduled, where the SF layer can be configured to perform any SF protocol and fulfil the specific control requirements passed down through Atomic-SDN.

Atomic-SDN has been designed to tackle the issues faced by current approaches to SDN in low-power wireless networks. It implements the three core services necessary for SDN control (*collection*, *configuration*, and *reaction*), as well as providing rapid and dynamic association with the SDN controller. Moreover, it facilitates these functions *as quickly as possible*, and as *as reliably as possible*. Depending on the network density, and the size of the network in terms of hop distance, it can maintains *scalability*. These services and their associated traffic patterns are detailed below.

- **Collection (*many-to-one*)**: Nodes need to be able to update the controller of their local and neighbourhood state, so that the controller can make informed decisions when configuring the network.
- **Configuration (*one-to-many*/*one-to-all*)**: The controller needs to be able to configure multiple nodes within the network, either to set data flows across the mesh, or to independently provide instruction to a number of nodes.
- **Reaction (*many-to-one*/*one-to-many*)**: Nodes need to be able to react to unexpected flows or events by soliciting the controller for instruction, and quickly receiving a response.
- **Association (*many-to-one*/*one-to-all*)**: Nodes need to be able to join the controller and be configured with initial instructions and network settings.

Atomic-SDN strays from the approach taken by previous works attempting to address the challenge of SDN architecture in low-power wireless networks. Rather than layering the SDN architecture on top of standard asynchronous or synchronous Layer-2 MAC options from the IEEE 802.15.4 networking stack, Atomic-SDN adopts SF as the mechanism for communication between the SDN controller and nodes within the multi-hop mesh network.

Indeed, SF is increasingly seen as the 'go-to' solution for low-latency control in low-power wireless networks, particularly when applications require highly-robust communication for unpredictable and opportunistic traffic patterns. This view is supported by the consistent and continued success of SF solutions in the IEEE EWSN Dependability Competition [7, 8, 97, 130, 131, DC2], which benchmarks protocols in terms of their reliability, latency, and energy efficiency across multi-hop networks.

5.3.1 General Approach

Atomic-SDN provides periodic SDN control *opportunities*: where an initial *indicator* (IND) flood, as shown in Figures 5.3, 5.9, and 5.10, instructs all network nodes as to the type of SDN service that will follow (if any), as well as maintaining time synchronisation across the mesh. This allows Atomic-SDN to separate SDN control from other network processes and slice the network across time so that control messages are no longer in contention with other protocols (such as RPL, 6LoWPAN, or application-layer). Due to the broadcast nature of SF, multiple nodes can be quickly and reliably serviced in a single flood, without replicating messages across multiple topology branches. This provides performance improvements that are orders-of-magnitude over current approaches to SDN in low-power wireless sensor networks.

However, to implement the different core SDN services within a multi-hop mesh network, multiple traffic patterns must be supported (*one-to-all*, *one-to-many*, *many-to-one*, *one-to-one*). Crucially, the plurality of these patterns are not supported by a single SF primitive or protocol and, as such, multiple protocols are needed to fulfil all required communication types. Unfortunately, the complex and low-level nature of SF implementations has meant that, until recently, there has been no unified framework allowing multiple SF protocols (such as Glossy [6], Chaos [100], LWB [99], or CRYSTAL [3]) to coexist within a single architecture.

Atomic-SDN solves this issue by introducing a novel SF architecture that allows the construction of complex, higher-level communication through the application of pre and post logic functions on top of SF primitives. In this manner, different flooding protocols can be configured, instantiated, and scheduled, as the SDN control requirements change; allowing Atomic-SDN to adapt the SF protocol to the requirements of the current SDN service as defined by the controller.

Figure 5.5 shows an overview of Atomic-SDN architecture in which, by applying configurable logic on top of generic flood primitives, SF control periods can be dynamically reconfigured to adapt to changing requirements from the SDN layer. This system is key to supporting the multiple different traffic patterns necessary to facilitate SDN in low-power wireless networks, and allows the simple implementation and configuration of multiple SF protocols that have been proposed to date, such as Glossy [6], CRYSTAL [3], or Low-Power Wireless Bus (LWB) [99]. The basic approach is as follows (moving upwards from the lower layers):

- The SF layer manages the lower level time synchronisation and concurrent transmissions.
- Floods are packaged into generic dedicated (single initiator) or shared (multiple initiators) flooding primitives.
- These primitives are configured with *offsets*, *guards*, and *protocol logic blocks* to create logical *phases*.
- Phases are linked and scheduled to create an SF protocol.

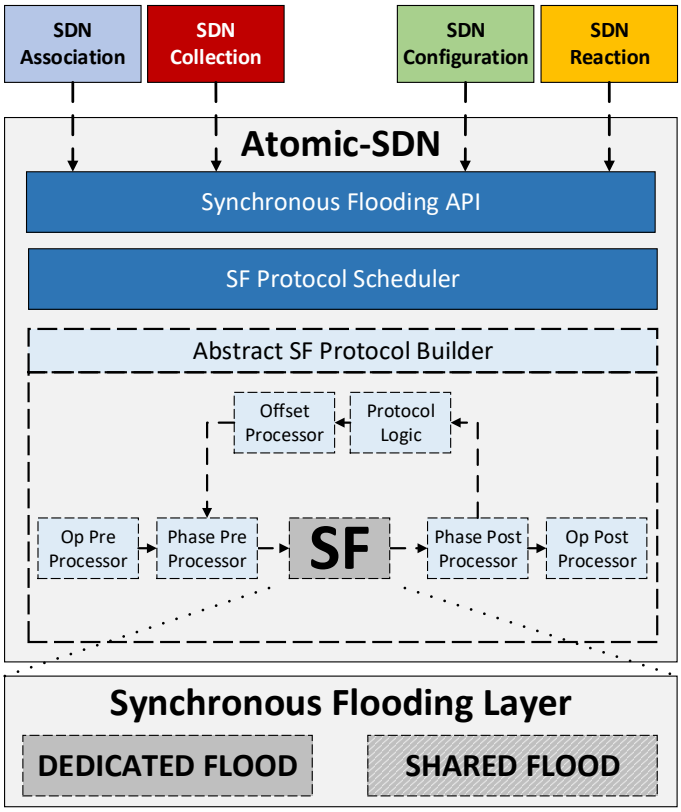


Figure 5.5: Atomic-SDN network stack. The abstract protocol middleware allows instantiation of concrete SF protocols through the application of pre and post processing logic on top of generic flood primitives. This mechanism allows Atomic-SDN to meet the complex traffic pattern requirements needed to facilitate SDN in low-power wireless networks.

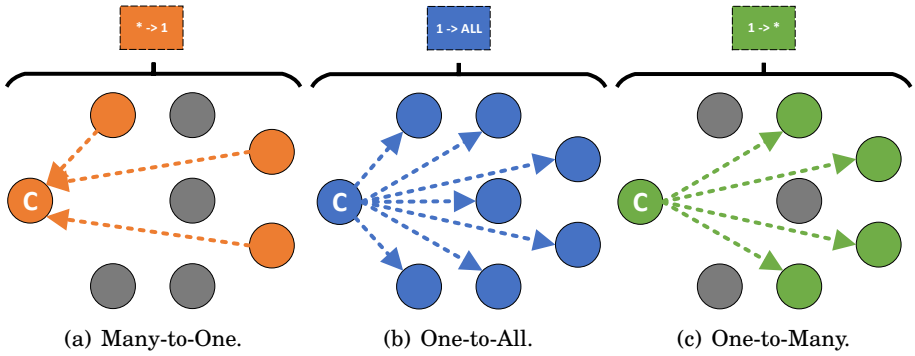
- SF protocols are mapped to SDN functions, and tailored to their current service requirements, to create an SDN control *opportunity*.
- The SDN controller periodically initiates a required SDN operation during a scheduled SF control slot.

5.3.2 Atomic-SDN Flooding Operations

To achieve the core SDN functions (*collection*, *configuration*, *reaction*), as well as facilitate extremely fast network *association*, Atomic-SDN needs to perform three distinct traffic patterns, as shown in Figure 5.6:

- Single source to all destinations (*one-to-all*)
- Single source to a subset of destinations (*one-to-many*)

- Multiple sources to a single destination (*many-to-one*)



(a) Many-to-One. (b) One-to-All. (c) One-to-Many.

Figure 5.6: Required traffic patterns for SDN control.

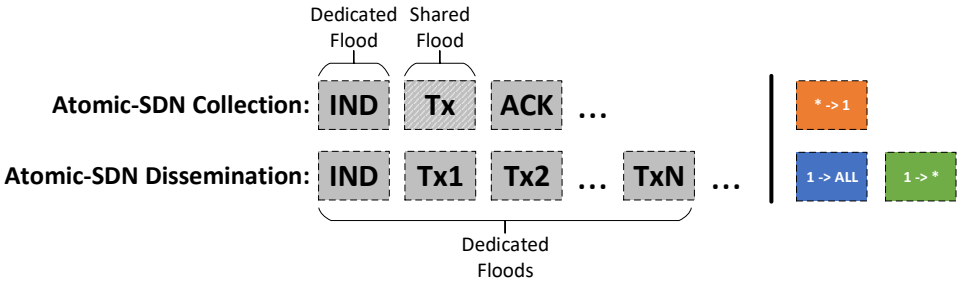


Figure 5.7: Atomic-SDN data collection and data dissemination protocols. Both protocols synchronise off an IND phase, before starting a series of shared or dedicated phases.

Atomic-SDN provides two SF protocols, (*collection* and *dissemination* shown in Figure 5.7), which can be used individually or in conjunction to fulfil these patterns. Each schedule then repeats until the SDN opportunity is complete. The first two traffic patterns (*one-to-all* and *one-to-many*) can be achieved through an SF *dissemination* flooding protocol. In its most simple case, this allows the controller to rapidly and reliably communicate information to the entire network within a single flood, allowing SDN to bypass the packet duplication issues inherent in other SDN architectures for low-power wireless. The flood is then propagated across the network as nodes successfully receive the packet and start to relay the transmission. If a node is assigned as a destination, it will read the packet data after the flood has ended, otherwise it will act as a forwarder.

The third pattern (*many-to-one*) is more complex. In SF collection protocols based on shared flood phases with different data, multiple sources will compete as initiators by relying on the capture effect [65]. In each flood, only one source will successfully be received by the destination. Therefore, competing nodes that were not successful must continue to retransmit until they are acknowledged in an ACK flood. As nodes are acknowledged they will switch their role from source to forwarder, and help with future transmission phases. Shown later in Figure 5.9, this continues

until all source nodes have had their transmissions acknowledged, which is indicated by a STOP consisting of one or more empty T_x floods plus a NACK phase.

5.3.3 Abstract Protocol Builder

The success of SF for control solutions in low-power wireless is rooted in the mechanism's ability to provide low-latency and high-reliability even under extremely adverse conditions. As such, there have been a number of attempts to take the core flooding principle, and tailor it to diverse application requirements in order to facilitate protocols for *one-to-all* communication [6], data collection [3], *many-to-many* communication [99], network consensus [100, 101], and interference management [97].

Each SF protocol satisfies a specific set of application requirements. However, a complete SDN architecture requires that a number of different traffic patterns be supported, and achieving this therefore requires multiple protocols. Yet the underlying low-level implementation of proposed SF protocols have, to date, varied significantly. Co-existence of multiple protocols within a single stack is particularly challenging despite being based on the same basic mechanism.

To address this issue, Atomic-SDN implements an Abstract Protocol Builder (APB) middleware layer (as shown in Figure 5.5), which uses generic flooding primitives attached with configurable protocol-specific logic to allow flexible construction of complex high-level synchronous flooding protocols. This mechanism is used within Atomic-SDN to implement the data collection and data dissemination protocols in Figure 5.7; however, the abstract nature of the APB means that it can be easily extended to implement any SF based protocol in order to suit additional traffic patterns or requirements.

5.3.3.1 Flood Primitives

In Atomic-SDN, generic SF *primitives* are defined as a single flood, configured with a MAX_TX number of transmission slots, each with duration T_{slot}. If a node is able to successfully complete all MAX_TX transmissions it will exit the flood process, otherwise it will exit at Δ_{SF}, the time taken for all transmission slots to elapse. Flood primitives are currently implemented as a *one-to-all* B2B T_x flood (Figure 2.23), however any lower synchronous flooding layer could conceivably be used, such as the Glossy interleaved R_xT_x approach [6] (Figure 2.21), or a consensus primitive such as Chaos [100].

5.3.3.2 Phases

Phases are the building blocks of Atomic-SDN, allowing higher-level SDN functionality to be realised by chaining multiple phases into a series of logic decisions. Each phase is a self-contained unit consisting of a flood primitive configured with MAX TX transmissions and duration Δ_{SF}, combined with an associated data structure and the concrete implementation of abstract func-

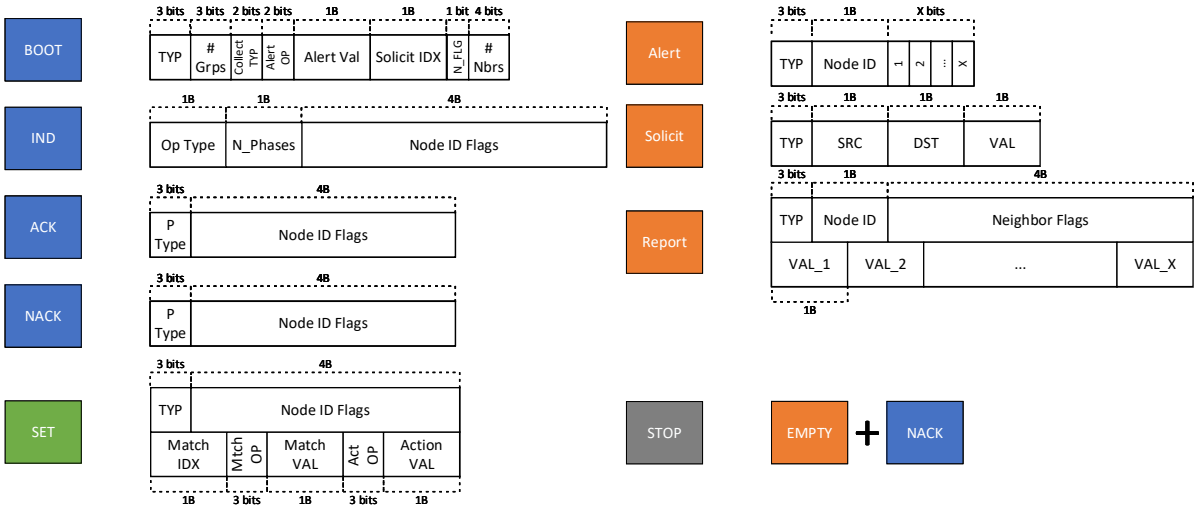


Figure 5.8: Atomic-SDN *phase* types built from the SF protocols outlined in Figure 5.7. These phases are chained together to create higher-level functionality in the form of an SDN 'opportunity'. From left to right: *one-to-all* phases (blue), *many-to-one* phases (orange), *one-to-many* phases (green), and a STOP phase (grey).

Table 5.2: Description of phases shown in Figure 5.8

Phase	**Description**
BOOT	Current SDN settings for network association.
IND	Indicate which *opportunity* type will follow (if any).
ACK	Acknowledge receptions at the controller.
NACK	Acknowledge no receptions at the controller.
SET	Configure an entry in the SDN flowtable.
ALERT	Alert the controller that an event has been triggered.
SOLICIT	Solicit the controller for instruction.
REPORT	Report state information to the controller.
STOP	End *opportunity* before the allotted time (Δ_{XOP}).

tions and configuration settings, shown in Figure 5.5, which define phase behaviour based on the current node role: *pre and post processing logic functions*, *guard times to allow for drift and processing in other nodes*, and *offsets from initial phase reference time*. Through these settings, phases can be configured to perform specific, self-contained roles, whilst propagating the associated phase packet types shown in Figure 5.8. Multiple phases can then be chained together in order to build up higher level processes, known as *opportunities*, allowing full protocols to be implemented through the combination of a number of simple blocks.

At the time of this thesis, the packets sent in each individual flooding phase are bounded by the Maximum Transmission Unit (MTU) of the underlying physical layer (127B in IEEE 802.15.4,

and 251B in). To send larger packet sizes a fragmentation and reassembly protocol would be required to split upper-layer MTUs across multiple floods. This could, for example, be achieved though selected elements and approaches of the 6LoWPAN protocol covered in Section 2.2.2. While this is an interesting avenue for further research, it is outwith the scope of this thesis.

5.3.3.3 Opportunities

Atomic-SDN defines the concept of SDN opportunities, whereby the controller regularly and synchronously initiates a period of SDN control across the network. These are shown in Figure 5.9, where highlighted phases blocks are repeated until the opportunity is complete; either through a predefined time limit, or through a STOP phase.

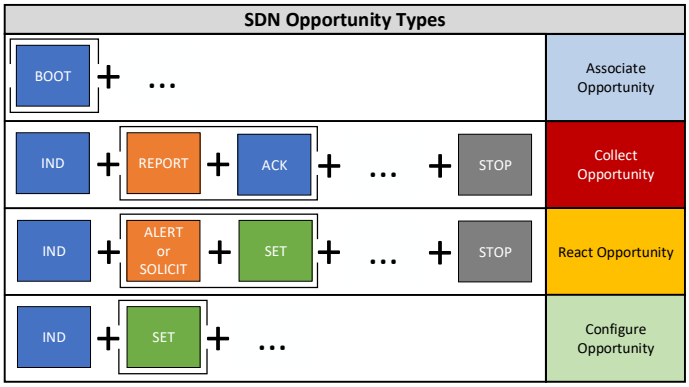


Figure 5.9: Atomic-SDN control *opportunities* built from the phase types defined in Figure 5.8. Highlighted phases are repeated until the end of the opportunity.

The type of opportunity is chosen by the controller prior to the flooding period, where the opportunity logic is constructed through the combination of a number of phase types, along with another round of pre and post processing logic. Prior to execution, every opportunity is announced by the controller through a special *one-to-many* Indicator (IND) phase. This phase instructs the network as to what type of SDN control opportunity to expect (if any), the number of phases in that opportunity, and distributes the current epoch sequence number. Additionally the IND phase includes a variable length array of mapped Node ID flags. Used in conjunction with the current opportunity type, these flags indicate the role of each node within the flood (*source*, *destination*, or *forwarder*).

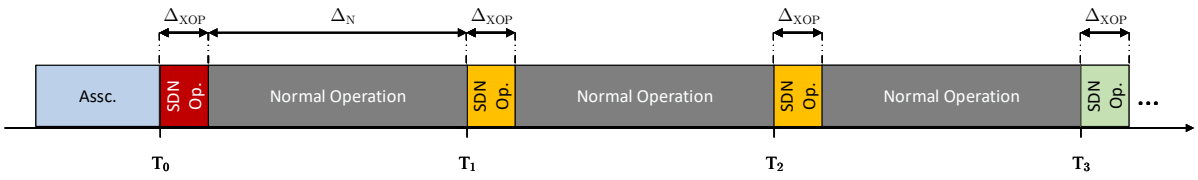
5.3.3.4 Epochs

An 'Epoch' is defined as the time between regularly scheduled SDN control opportunities, with periodicity T_i, which dictates the frequency of SDN opportunities. As synchronous flooding periods in Atomic-SDN inherently block other processes, a longer epoch devotes a greater proportion

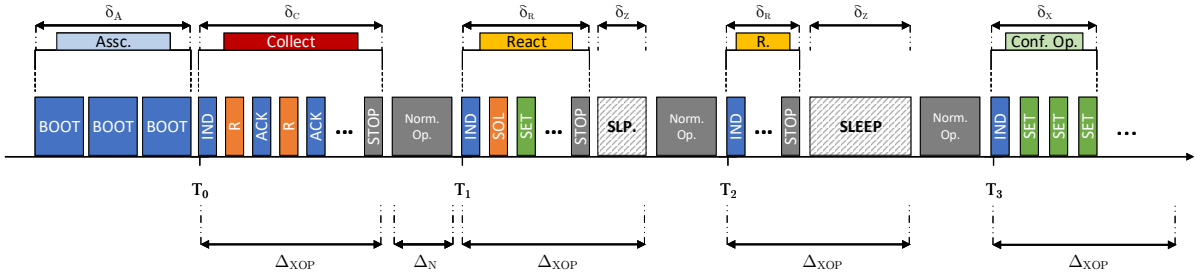
of time to normal network operation; whether that is application processes, other low-power wireless protocols, or to allow nodes to sleep and therefore conserve energy.

5.3.4 Scheduling

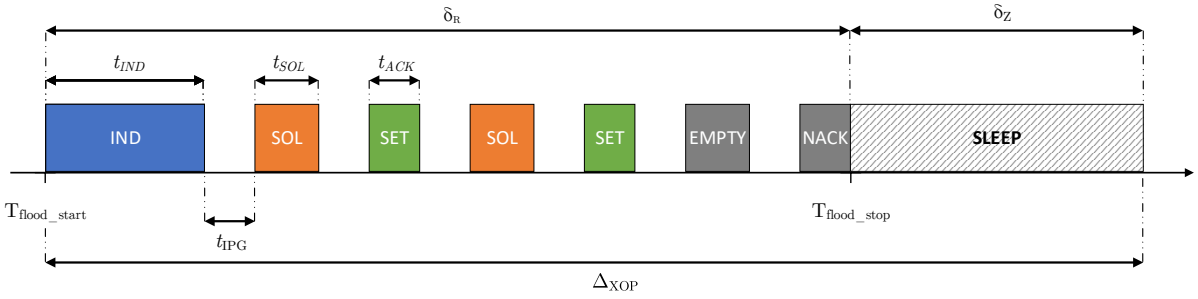
Atomic-SDN operates a two-stage scheduling process, as highlighted in Figure 5.10. Firstly, self-contained flood 'Phases' are chained together within a short period to allow the construction of higher-level SDN functionality. Then, at a macro level, these flooding periods are scheduled periodically to provide regular SDN 'opportunities', as well as maintaining tight time synchronisation across all nodes. With reference to Figure 5.10, the following subsections explore these two distinct stages.



(a) Atomic-SDN epoch schedule.



(b) Atomic-SDN opportunity schedule.



(c) Atomic-SDN phase schedule (REACT Opportunity).

Figure 5.10: Example Atomic-SDN schedule at epoch, opportunity, and phase level..

5.3.4.1 High-Level 'Opportunity' Scheduling

One of the core principles behind Atomic-SDN is the separation in time of control processes from normal network operation. By slicing control independently from normal operation, the controller

is able to define a short period in which it is able to communicate with and instruct associated nodes in the local mesh.

With each control period serving a single SDN control function, this necessitates some decision making and scheduling from the controller: choosing what type of control opportunity to initiate at the start of each epoch, and instructing nodes when to quit the control period and resume normal network operation. This scheduling process is shown in Figure 5.10a which shows a high-level timeline of Atomic-SDN.

Once the type of SDN control opportunity is chosen, a mandatory *indicator* (IND) phase is scheduled at the start of the control period. This *one-to-many* phase allows the controller to propagate the opportunity type (if any) to the rest of the network, as well as assigning nodes' roles (*source*, *destination*, or *forwarder*) and distributing any additional information, such as maximum length of the control period.

5.3.4.2 Low-Level 'Phase' Scheduling

Each flood is packaged into self-contained 'phases' which accomplish specific functions within a larger SDN opportunity. Figure 5.10b shows how, after receiving the IND phase propagated at the start of each opportunity, nodes within the mesh participate in a predefined schedule mapped to the SDN opportunity defined within the IND. This schedule consists of a number of distinct phases of one or more types, and each phase (in and of itself) has its own low-level slot schedule. The phase schedule is configured depending on the current phase type, where guard times, offsets, and protocol logic are determined from the current node role within the context of the larger SDN opportunity. Figure 5.10c shows an example of a *react* opportunity phase schedule, where the IND phase is preceded by SOL/SET phase pairs that repeat until all participating nodes have completed the opportunity.

5.3.5 Channel Hopping and Network Association

Network association is achieved through BOOT and IND phases. BOOT phases distribute the current SDN configuration to joining nodes (match/action information for flowtables, what information should be included in *collect* opportunities, etc.). IND phases are scheduled every epoch and, as well as containing SDN opportunity information, allow nodes to reassociate themselves if they have de-synchronised from the network

Figure 5.11 shows how Atomic-SDN employs per-slot channel hopping as proposed in [7]. In every IND or BOOT phase the Atomic-SDN controller distributes the current epoch sequence number, which is used to generate a pseudo-random channel hopping sequence for the network. Once known, nodes increment this number every epoch, meaning that if they miss an IND phase due to their duty-cycling or from interference they will retain knowledge of the hopping sequence. At SF primitive slot, nodes concurrently hop to the next channel in this sequence. Known association channels are seeded into every second channel in the sequence (for example,

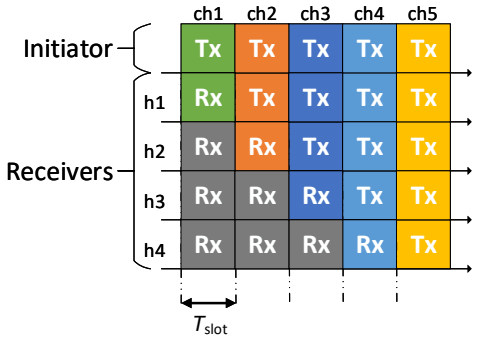


Figure 5.11: Per-slot channel hopping in an Atomic-SDN flood.

$\{ch2, ch3\}$), so that when a node is trying to join (or rejoin) the network, it merely has to listen on one of these known channels for a long enough period until it hears a transmission and can resynchronise to the controller.

This mechanism, combined with the spatial diversity of SF, allows Atomic-SDN to survive extremely high levels of interference compared to other SDN architectures for low-power wireless networks, providing reliable network control where other solutions would struggle. It has successfully implemented and tested within competition scenarios [136], and is further evaluated in the next section.

5.4 Evaluating Atomic-SDN

This section evaluates Atomic-SDN: comparing performance metrics against other SDN implementations for low-power wireless networks, the time taken for nodes to associate with the SDN controller, and scalability (degradation in performance) as the size of the mesh network increases. μSDN and SDN-WISE were used to provide a benchmark against multiple different MAC layers for IEEE 802.15.4. Results show that Atomic-SDN displays considerable and extensive performance gains across all metrics.

5.4.1 Simulation Results

Atomic-SDN performance is compared against the μSDN [C1] architecture from Chapters 3 and 4; and SDN-WISE [26], a SDN implementation for IEEE 802.15.4 based on the RIME communication stack [117]. Both are implemented in Contiki [122], the same low power Operating System (OS) on which Atomic-SDN is built. Both were chosen evaluation candidates as they can run on top of multiple different IEEE 802.15.4 MAC layers, and therefore provide multiple baselines to compare Atomic-SDN performance. This section demonstrates that, by utilising SF to create periodic SDN control slices, Atomic-SDN displays considerable performance gains

across all metrics in comparison to other SDN architectures for low-power wireless. Furthermore, this mechanism is only possible due to the novel SF framework developed for Atomic-SDN, which allows multiple SF protocols to be configured and instantiated in order to satisfy the plurality of traffic patterns necessary for full SDN control. Figures [5.12 - 5.15] summarise the results.

5.4.1.1 Simulation Setup

All simulation configuration settings are outlined in Table 5.3. Simulations were performed using Cooja, a simulator and hardware emulator for Contiki OS [122]. SF primitives require specific radio drivers to be written for target hardware and, at the time of development, the B2B T_x primitive employed by Atomic-SDN was written specifically for [DC2] and so was only supported by the CM5000 TelosB[1] platform boasting a TI MSP430F1611 CPU and CC2420 radio. Conveniently, this platform is also supported by SDN-WISE; and Cooja also emulates the EXP5438 motes (TI MSP430F5438 CPU and CC2420 radio) needed by a previous version of μSDN due to memory requirements. Additionally, Cooja provides a simulated Multipath Ray-tracer Medium (MRM) radio environment that allows rays to be combined at the receiver (necessary to simulate CT). This ray-tracing model differs from the simple UDGM model used in Chapters 4 and 5, in that it allows transmissions from multiple sources to be combined at the receiver, a necessary fact of concurrent transmissions. While this does not address all physical layer phenomena (in particular it doesn't consider beating effects resulting from relative CFO differences between two transmitters) it does allow rough approximation of how CT-based flooding performs at the MAC layer. All simulations were performed across a grid topology, with nodes placed at 300m intervals.

As the Atomic-SDN controller needs to keep track of all network nodes, a maximum of 70 nodes were used due to the memory constraints of the emulated TelosB hardware. Simulations to evaluate performance were run over a 1h period, with an SDN opportunity frequency of 1 second. At the start of each opportunity the type of SDN control scenario (either *collection*, *configuration*, or *reaction*) was set in a round-robin process. Atomic-SDN was evaluated against μSDN and SDN-WISE, which were both configured to adopt two separate MAC scenarios: firstly ContikiMAC, an energy saving MAC layer, and secondly *always-on* Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA). In each simulation the controller collects state information from all nodes every 60s, and node flowtable entries have a 300s lifetime. Additionally, each simulation runs a data collection application where nodes attempt to send application data to a sink node at a variable rate of 60→75s.

5.4.1.2 Simulation: SDN Scalability in the Mesh

A key challenge for SDN in IEEE 802.15.4 low-power wireless has been to maintain scalability as the mesh grows from a handful to hundreds of nodes within a local cluster. As the number

[1] https://www.advanticsys.com/shop/mtmcm5000msp-p-14.html

Table 5.3: Simulation Parameters

Parameter	**Setting**
Duration	1h
Topology	Grid
Radio Medium	Multipath Ray-tracer Medium (MRM)
Link PRR	~90%
Transmission Range	~300m
SNR Reception Threshold	-100dB
Capture Effect Preamble	64μs
Capture Effect Threshold	3dB
SDN *Collect* Period	60s
SDN Flowtable Lifetime	300s
Application Data Period	60→75s

of nodes increases, the controller needs to appropriate a greater proportion of limited network resources to SDN control.

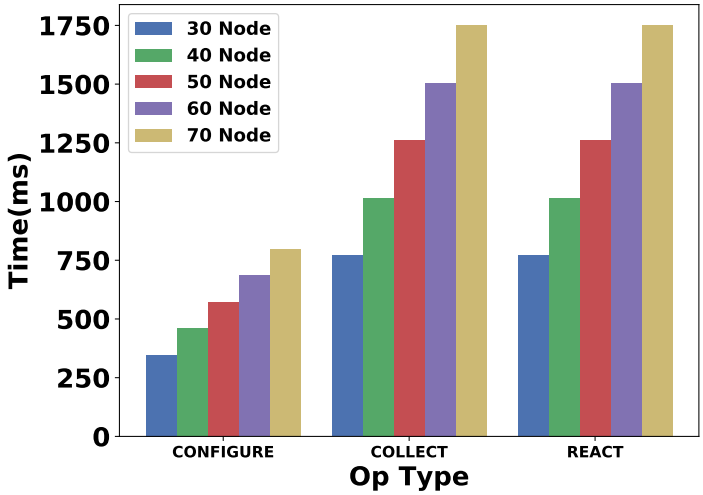


Figure 5.12: Time taken to complete each Atomic-SDN opportunity as the local mesh scales.

This is explored through the evaluation of the completion time for the three SDN opportunity types (*collection*, *configuration*, and *reaction*) in networks of increasing size: running simulations for 30, 40, 50, 60, and 70 node mesh networks (limited to 70 nodes due to hardware memory constraints at the Atomic-SDN controller node). Figure 5.12 shows these results, and demonstrates how the use of SF protocols means that the time to complete all SDN opportunity types (for all network nodes) increases linearly with network size, regardless of hop count.

In each SDN opportunity type a worst-case scenario has been assumed, where the controller needs to interact with each node independently. However the number of nodes participating in each SDN opportunity, necessary to fulfil the requirements of higher-level application functions (virtually located at a centralised controller), would likely be a smaller subset of all nodes and therefore incur lower delay.

The SDN *configure* opportunity is a *one-to-all* process where, after the initial indication phase, each node is configured in turn, in a scheduled fashion. In a 70 node network this allows the configuration of all network nodes within 800ms, assuming each node requires a separate configuration message. However, this time could be substantially reduced (to tens of milliseconds) if a configuration message is relevant to all, or a subset of nodes; allowing the SDN controller to configure multiple nodes in a single flooding phase.

Both the *collect* and *react* opportunities utilise the same underlying SF protocol, and therefore exhibit equivalent delay. In this two phase protocol, the competition between nodes to successfully transmit their data to the controller means that the minimum bound on the completion time is dictated by the number of nodes that need to communicate with the controller. This is a worst-case scenario where it is assumed that all nodes try to perform the SDN operation at exactly the same time, which inevitably causes contention and retransmissions. It does not necessarily follow that this would be the case in a real-world situation. However, these times are still considerably less than the time it takes to complete the same SDN operations in current SDN architectures for low-power wireless networks, as demonstrated later in Figure 5.14a.

Yet, despite Atomic-SDN demonstrating considerable scalability in comparison to other low-power wireless SDN architectures, there are still questions surrounding the scalability of Concurrent Transmissions (CT) when there are 100s or 1000s of nodes and a large number of hops [137]; particularly in extremely dense networks where the number of concurrent transmitters could be significant. Although the authors of [138] propose mechanisms for managing these issues and employing CT based protocols across large networks, there has yet to be experimental evaluation of large-scale SF in real-world scenarios.

5.4.1.3 Simulation: Performance Comparison

Atomic-SDN is compared against current approaches to SDN in IEEE 802.15.4 using μSDN and SDN-WISE in two Layer-2 configurations: always-on CSMA and the duty-cycled ContikiMAC. Simulations were performed in Cooja using emulated target hardware, and Atomic-SDN performance gains are evaluated in terms of *latency*, *reliability*, and *energy efficiency*. The simulated network was limited to 30 nodes in order to accommodate SDN-WISE which, although specified as able to support longer routing headers [26], does not have this feature available in the current SDN-WISE Contiki implementation.

'Best-Case' - Individual Node Participates: First a 'best-case' scenario is considered, where just a single node participates in the SDN control operation (i.e. there is no competing SDN

Table 5.4: Mean latency, Packet Delivery Ratio (PDR), and Radio Duty Cycle (RDC) for a single node to perform each SDN opportunity type in a 30 node network.

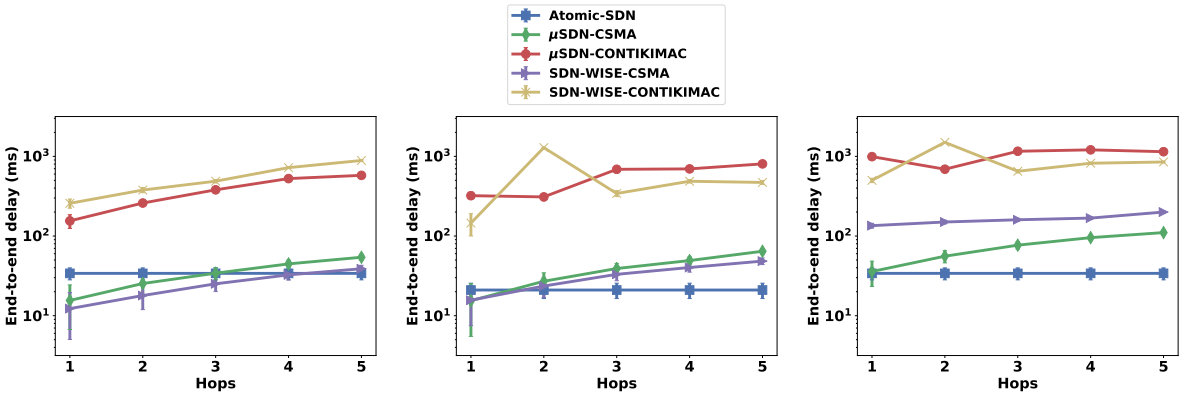
Architecture	**Latency (ms)**	**PDR (%)**	**RDC (%)**
Atomic-SDN	34.0	100.00	1.34
μSDN-CSMA	33.94	99.58	100
μSDN-ContikiMAC	340.42	96.45	3.76
SDN-WISE-CSMA	25.75	68.49	100
SDN-WISE-ContikiMAC	544.23	93.84	5.15

control traffic from other nodes). Table 5.4 averages these results and Figure 5.13 shows mean delay versus hop distance from the SDN controller for an individual node performing one of three SDN operations: *collection*, *configuration*, and *reaction*. In each case, Atomic-SDN maintains consistent delay over all distances, due to the minimum bounds on latency inherent in SF protocols. However, as expanded upon in the previous section, this bound is affected by the configuration of the underlying SF protocol that supports it (which can require N number of phases). In this particular scenario, where there is only one individual node communicating with the controller, it is possible for CSMA based architectures to achieve better latency results than Atomic-SDN at lower hop distances depending on the configuration of the lower layer SF primitive (guard times, number of slots, etc.). However, in CSMA, the radio is always on. Compared to the duty-cycled ContikiMAC configurations of μSDN and SDN-WISE, Atomic-SDN maintains minimal latency bound by the length of the flood, regardless of hop distance.

'Worst-Case' - All Nodes Participate: Next performance is evaluated when considering a 'worst-case' scenario, where all nodes in the network need to participate in the SDN control operation simultaneously. These results are presented in Figure 5.14.

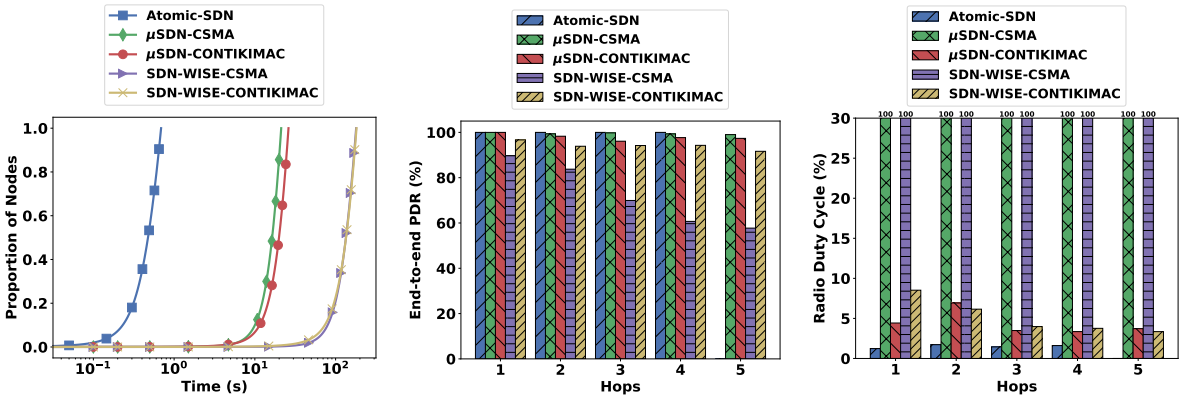
The SDN *react* operation is used to benchmark the time taken for all network nodes to concurrently solicit and receive a single instruction from the controller. Figure 5.14a shows the effectiveness of SF protocols in comparison to current SDN implementations for low-power wireless, where nodes need to perform two or three-way handshakes across multiple Layer 2/3 links. Not only is Atomic-SDN able to perform this operation on all network nodes within milliseconds, but this is orders-of-magnitude faster than the other SDN approaches, which can take seconds or even minutes.

Figure 5.14b shows Atomic-SDN achieves far higher reliability compared to both the CSMA and ContikiMAC configurations of μSDN and SDN-WISE. As μSDN implements end-to-end acknowledgements for SDN control traffic, it presents with a higher Packet Delivery Ratio (PDR) in comparison to SDN-WISE (particularly the CSMA configuration) which has no transportation layer guarantees. Additionally, ContikiMAC causes high channel utilisation through packet retransmissions. Although this improves the overall PDR in the case of SDN-WISE, as packets have a higher chance of surviving each hop, it also causes considerable contention on links that experience high traffic loads. This therefore results in a drop in PDR for μSDN nodes at greater



(a) Mean collect delay. (b) Mean configure delay. (c) Mean react delay.

Figure 5.13: Mean collection, configuration, and reaction delays versus hop distance in 30 node network when an individual node participates. Atomic-SDN exhibits similar latencies to CSMA-based μSDN and SDN-WISE, however it additionally retains consistent delay across all hop counts.



(a) Time taken for an SDN *react* operation for all nodes. (b) Mean SDN control traffic PDR for all nodes. (c) Mean RDC over hop distance for all nodes.

Figure 5.14: (a) Time taken to complete an SDN *react* operation concurrently for all nodes in a 30 node network, as well as (b) end-to-end Packet Delivery Ratio (PDR), and (c) Radio Duty Cycling (RDC) versus hop distance from the controller. As CSMA based μSDN and SDN-WISE are always-on, they exhibit 100% RDC, denoted at the top of the plot.

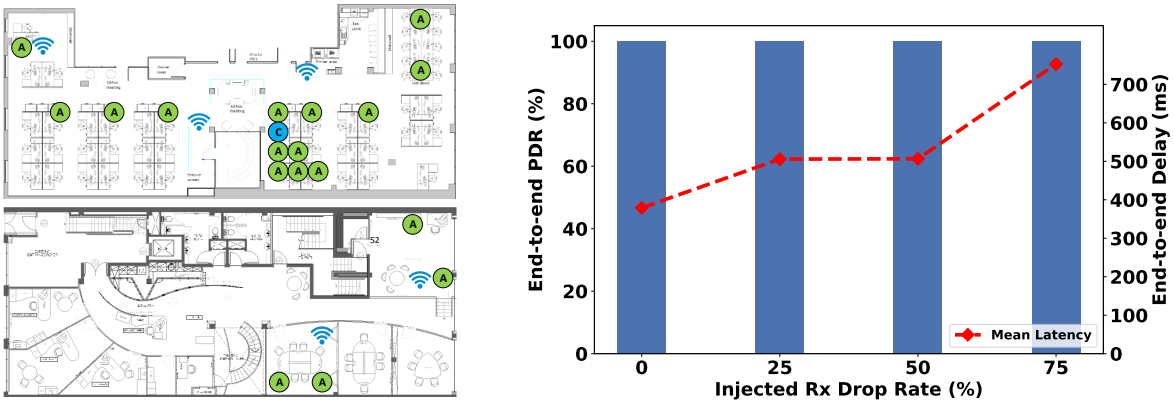
hop distances as they contend with nodes nearer the controller.

Finally, Figure 5.14c shows the Radio Duty Cycling (RDC) at each hop. ContikiMAC configurations show reduced energy efficiency at lower hop counts as nodes need to serve messages from their children; whilst energy for Atomic-SDN increases at higher hop counts as nodes closer to the controller receive in the first few transmission slots on an ACK, and so spend less time participating in the flood. CSMA configurations, which don't perform any duty-cycling, display high PDR across all hops. Barring any contention, nodes should always be able to receive

transmissions as the radio is always on. In comparison, the use of SF in Atomic-SDN means it can benefit from a highly reliable MAC, whilst retaining the low-energy operation of duty-cycled approaches. By using the APB to construct multiple SF protocols tailored for each SDN task Atomic-SDN demonstrates near perfect reliability when collecting state information from all nodes. Although these results are based on simulation and 100% reliability cannot be guaranteed, multiple studies and extensive experimental evaluations (as covered in a recent survey [94]) have demonstrated up to 99.99% reliability is achievable using SF protocols, even under heavy interference.

5.4.2 Testbed: SDN Control Under Interference

A 19 node testbed was used to evaluate the capability of Atomic-SDN to provide reliable SDN control in high interference scenarios, reaching all mesh nodes irrespective of link quality. The layout of this testbed is shown in Figure 5.15a. Nodes are located over two floors and it presents a number of interesting features such as a dense cluster, isolated multi-hop paths, and non Line-of-Sight (LOS) lossy links.



(a) 19 Node Atomic-SDN Testbed.

(b) *React* PDR and latency versus Rx misses.

Figure 5.15: (a) Evaluation of Atomic-SDN on a 19 node TelosB testbed at the Toshiba Bristol Research and Innovation Laboratory (BRIL). Initiating nodes are marked in green, and the SDN controller in blue. (b) PDR overlayed with mean round-trip latency for SDN *react* operations injected with probabilistic reception misses, in a 19 node testbed. Atomic-SDN maintains high-reliability even during 75% injection of receive misses.

The SDN controller was configured to repeatedly initiate an SDN *reaction* operation from all nodes with minimal periodicity (i.e. back-to-back) over 1h periods. Forced reception (R_x) drops were injected into the SF layer at each node in order to simulate the effects of heavy external interference, stepping from 0%, 25%, 50% to 75% probability. Latency and PDR results for each drop rate are presented in Figure 5.15b, and show how Atomic-SDN is able to maintain near 100% probability even with 75% reception misses. However, it shows that as the probability

of missing a reception increases, this also increases the average time taken for each node to complete the SDN operation. As receptions are missed, the underlying SF protocol ensures nodes will retransmit their controller solicitation data until they hear an acknowledgement.

Not only do these testbed results demonstrate the resilience of the SF approach to SDN control pioneered in Atomic-SDN (and also shown through success at the IEEE EWSN Dependability Competition), but overcoming this issue of resilience and reliability is particularly relevant in both Advanced Metering Infrastructure (AMI) and industrial control scenarios: where nodes may suffer from lossy links due to harsh environments, or multipath effects due to surrounding industrial machinery and assets. By maintaining a reliable and low-latency link to an SDN controller, based on a stateless and topology agnostic control mechanism, Atomic-SDN can provide programmable SDN control without assuming network stability.

5.5 IEEE EWSN Dependability Competition

Atomic-SDN's Abstract Protocol Builder (APB) was used to address the considerable and varied challenges of the EWSN 2019 Dependability Competition, in an Adaptive Software Defined Scheduling (ASDS) approach based on the Atomic-SDN architecture. This provided a flexible SF solution that could instantiate, tailor, and schedule multiple different protocols based on information gathered in an initial configuration phase. While this thesis is supported by two competition entries, CROWN (2018) [DC1] and ASDS (2019) [DC2], this section only considers [DC2], which won runner-up in both categories of the competition against 11 other teams, was the only entry to place in both categories, and was the only solution to achieve 100% reliability at the highest interference level.

Section 5.5.1 provides a brief outline of the competition scenarios, while Sections 5.5.2 and 5.5.3 cover the high-level protocols and low-level optimisations used in the competition. Finally, Section 5.5.4 presents a brief summary of competition results for each scenario. Full results and details of the competition are made available online [29], while a selected subset of results are included in Appendix A of this thesis[2].

5.5.1 Competition Scenario

The IEEE EWSN Dependability Competition was started in 2016 by organisers at the Graz University of Technology (TU Graz) in Austria, and invites teams to benchmark solutions on a multi-hop testbed against increasing levels of interference. This was partly as an attempt to address the considerable variation across benchmarking of low-power wireless solutions, where much of the research of the past few decades has been performed over various testbeds, simulators, and hardware platforms, and there are legitimate questions either towards its reproducibility, or comparison against other solutions. Remote participation in the competition has been in place

[2]This author has obtained permission from the competition organisers to reproduce these results in this thesis.

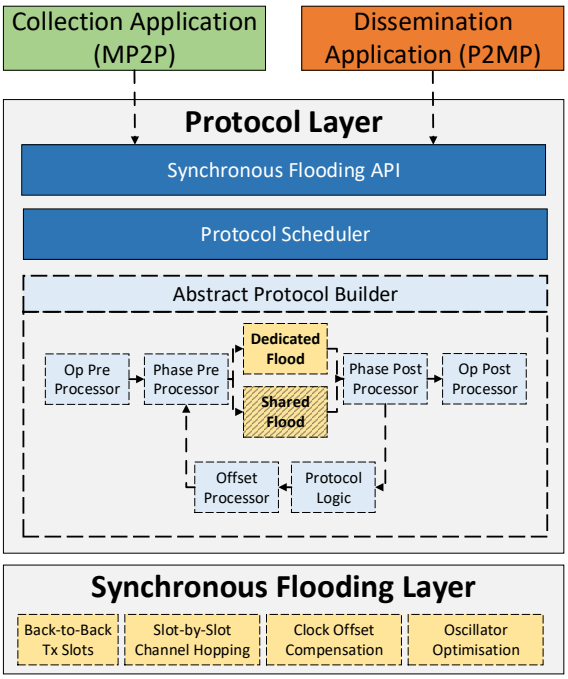


Figure 5.16: ASDS stack modified from the Atomic-SDN architecture.

since 2018 (as opposed to a 'hackathon' format in 2016 and 2017), allowing teams to test their solutions up to two months before requiring upload of the final binary.

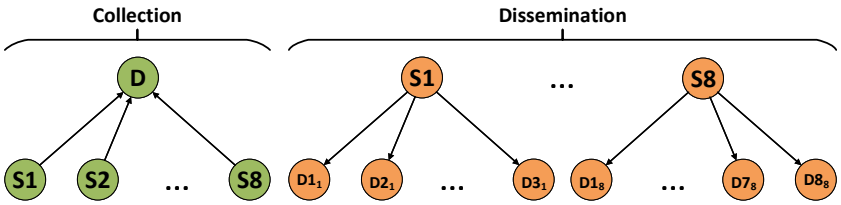


Figure 5.17: 2019 IEEE EWSN Dependability Competition *collection* and *dissemination* scenarios.

The competition was held in two separate categories, as shown in Figure 5.17. Firstly, a *data collection* scenario, where up to eight source nodes communicate to a single destination node over a multi-hop network (*multipoint-to-point* (MP2P) traffic). Secondly, a *data dissemination* scenario, where up to eight source nodes disseminate actuation commands to a unique set of destinations nodes in the network (*point-to-multipoint* (P2MP) traffic).

$$T = E \times K_E + L \times K_L + R \times K_R \tag{5.1}$$

Entries were evaluated over multiple runs in terms of *reliability*, *latency*, and *energy efficiency*, where score for each scenario is determined by a single metric, T in Equation 5.1. In this calculation, E is the relative energy, L is the relative Latency, R is the relative reliability, and

$K_E = K_L = 1$ and $K_R = 20$ weight the three metrics. Moreover, the 2019 competition considerably increased the complexity of the task by introducing a host of new scenarios, meaning that solutions were tested over 72 different permutations in total.

- 3 data lengths (8, 32, and 64 bytes).
- 3 different traffic loads (*periodic* 5s/30s and *aperiodic*).
- Varying *source* and *destination* nodes.
- 2 different node layouts with both *sparse* and *dense* areas.
- 4 jamming levels: *none*, *mild*, *strong*, and *dynamic*.

Given the complexity and variation in the possible scenarios, the solution was built on top of the Atomic-SDN APB, which provided a single framework that could adapt protocol behaviour by configuring high-level logic on top of lower layer synchronous flooding primitives in order to meet the traffic requirements in each specific test. Figure 5.16 shows how the Atomic-SDN stack was modified.

5.5.2 Protocol Operation

The Atomic-SDN APB, was used alongside lower layer SF primitive optimisations to construct protocols (Figure 5.18) capable of handling both periodic and aperiodic data in both of the competition scenarios.

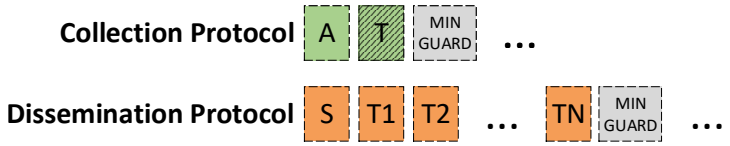


Figure 5.18: Phase-level depiction of ASDS collection and dissemination protocols. The collection protocol differs from the Atomic-SDN base collection protocol in that it eschews the IND phase and synchronises off an ACK.

In both protocols, the periodicity between each round of the protocol operation was calculated as the accumulative duration of each phase, plus the minimum guard time needed to either read or write a packet from the EEPROM, as shown in Figure 5.19.

Collection Protocol: A *2-phase* flooding operation was used in the collection protocol, consisting of a dedicated acknowledgement (ACK) phase and a shared transmission (TX) phase. In each round of the protocol operation, sources compete in the TX phase to try and send their data to the destination, while the destination uses the ACK phase to acknowledge any sources heard in the previous round. For purposes of maintaining network synchronisation, the ACK also serves as the synchronisation phase, meaning the destination must act as the network timesync.

Figure 5.19: Timeline of back-to-back protocol operation, with short guard times to ensure minimal latency.

Dissemination Protocol: An *N-phase* flooding operation was used in the dissemination protocol. An initial synchronisation phase allows any node to serve as the network timesync, while all sources are scheduled a dedicated phase in which to act as the flood initiator. Sources will repeatedly transmit their last packet until the arrival of new data.

5.5.3 Low-Level Optimisation

In addition to the higher layer protocols constructed through the Atomic-SDN protocol builder, there was extensive optimisation of the synchronous flooding layer, based on the particular idiosyncrasies of the MSP430 and CC2420 hardware used within the competition.

- *Back-to-back transmissions*: First proposed in the 2017 competition [7], ASDS performs back-to-back transmissions after a node first successfully receives. This allows a packet to rapidly propagate across the network, as opposed to an interleaving Rx-Tx model.
- *Slot-by-slot random channel hopping*: A pseudo-random array is generated before every protocol operation, allowing the synchronous flooding layer to hop to a different channel in every slot. Due to the spatially diverse and high levels of interference experienced in the competition, this rapid channel hopping allows floods to survive propagation across the entire network.
- *Clock-offset estimation*: We employ on-the-fly clock offset compensation [28] on the unstable MSP430 oscillator in order to mitigate the clock frequency deviations across the multi-hop network. This helps to maintain the 0.5μs synchronisation between nodes that is necessary to take advantage of concurrent transmissions.
- *Radio oscillator optimisation*: If the period between finishing a flooding phase and starting the next allows time to turn off and then turn on the oscillator, then the radio is turned off in order to save energy.
- *Number of transmissions*: As the radio has already ramped up for transmission by the time a CRC check fails, it is necessary to skip the next Tx slot if the packet was corrupted. This, alongside a high network hop count and completely jammed channels, necessitates a high number of transmission slots in each phase.

5.5.4 Testbed Evaluation and Benchmarking

The reader should note that results provided in this section were generated as part of the IEEE EWSN 2019 Dependability Competition. The benchmarking was therefore run by the competition organisers at the Graz University of Technology, and results that evaluate and compare solutions other than the author's own submission to the competition are therefore not a direct contribution towards this PhD. However, they are included in this section in order to provide the reader with empirical evaluation of SF protocols in general, and demonstrate how there is no one SF protocol that is able to satisfy all QoS requirements for all application scenarios. A flexible SF solution based on the Atomic-SDN approach, such as the ASDS entry submitted by the author, can not only handle multiple scenarios, but can provide QoS guarantees under the sort of external interference commonly encountered in real-world scenarios: a claim supported by these benchmarking results.

5.5.4.1 Data Collection Category

Table 5.5: Median results for data collection category, for both periodic and aperiodic data, all jamming scenarios, and all packet sizes. 6TiSCH [2] and CRYSTAL [3] are included as baselines for comparison.

Team	Rank in Category	Energy(J)	Reliability(%)	Latency(ms)
01	1	1974.69	100.00	422.86
06 (ASDS)	2	1818.82	100.00	317.90
10	3	1975.30	97.02	315.23
12 (6TiSCH)	4	1154.83	64.31	793.10
11 (CRYSTAL)	5	1331.76	97.53	1528.88
03	6	682.37	55.20	1235.72

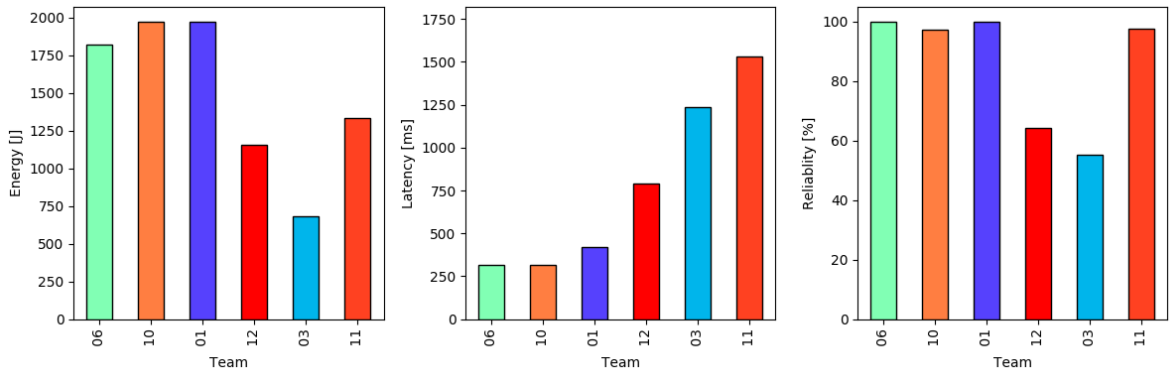


Figure 5.20: Summary of data collection category results from Table 5.5

5.5.4.2 Data Dissemination Category

Table 5.6: Median results for data dissemination category, for both periodic and aperiodic data, all jamming scenarios, and all packet sizes. CRYSTAL [3] is included as a baseline for comparison.

Team	Rank in Category	Energy(J)	Reliability(%)	Latency(ms)
10	1	1488.69	91.01	196.33
06 (ASDS)	2	1728.84	91.37	556.96
11 (CRYSTAL)	3	1628.47	94.44	1388.07
08	4	1555.41	43.20	1372.36
03	5	447.59	30.89	3054.77

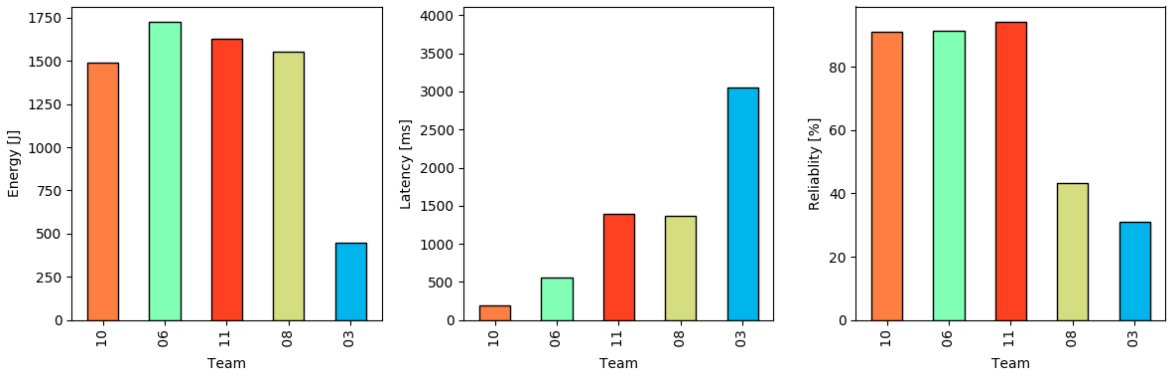


Figure 5.21: Summary of data collection category results from Table 5.6

5.6 Summary and Conclusions

This chapter has introduced the design and implementation of Atomic-SDN, a unique solution for SDN in low-power wireless networks that utilises Synchronous Flooding (SF) to provide low-latency, reliable SDN control. By facilitating the propagation of control messages across the network using SF, Atomic-SDN allows the SDN control plane to operate without knowledge of network topology and communicate at theoretical bounds of minimal latency. Furthermore, the aggressive spatial, temporal, and frequency diversity inherent within SF protocols means Atomic-SDN is extremely reliable and resilient to both poor link quality as well as external interference.

To satisfy the complex traffic pattern requirements needed to facilitate the various SDN services, a novel architecture was proposed, and has since been submitted as patent application [P1]. Using this approach, multiple synchronous flooding protocols can be configured and tailored

to the application requirements. This is achieved by reducing individual floods to base *primitives* and using these blocks to build higher-level functionality as SDN control *opportunities*. Within regular control timeslots, Atomic-SDN can perform any SDN service in a fraction of the time required by current SOTA low-power wireless SDN architectures, and in a single flood can simultaneously do this for multiple nodes.

Atomic-SDN has been implemented in Contiki [122] for TelosB (TI MSP430F5438 CPU and CC2420 radio) hardware, and evaluated through both simulation and testbed experimentation. Furthermore, it has been demonstrably shown to display resilience to extremely high interference, not only through the testbed experimentation in this chapter, but also in extensive benchmarking against WiFi [123] interference at the 2019 IEEE EWSN Dependability Competition [DC2].

5.6.1 Research Questions

This chapter has presented material supported by publications [C3, J1], and patent application [P1]. Through the development and implementation of a novel SF-based control architecture, Atomic-SDN displays some key advantages over current SDN architectures that employ standard MAC approaches, which addresses research questions [Q2, Q3, Q4].

- Temporal decoupling means *control signalling can be isolated* from other processes. [Q2]
- SF can satisfy *one-to-many*/*one-to-all* SDN traffic requirements. [Q3, Q4]
- Aggressive temporal, spatial, and frequency diversity provide *high-reliability*. [Q3, Q4]
- Communication is facilitated at theoretical *minimum bounds of latency*. [Q4]

[Q2]: *How does SDN control signalling affect the low-power wireless mesh?* Though chapter 2 explored the concept of Layer-2 isolation in order to decouple SDN control and not impede on other traffic flows, a major weakness in this solution lies in how to provision and guarantee resources for all nodes. Yet while such an approach is limited by the resources available in the slotframe, SF not only allows Atomic-SDN to both communicate with all nodes in a single flood, but it can bound SDN control periods within a maximum allotted time. This allows the SDN controller to support guarantees on the proportion of network resources that are provisioned for SDN control signalling. Although Atomic-SDN consequently monopolises all network resources during any given flooding period, this approach eliminates contention with other processes. In constrast to other recent works, which try to distribute control and reduce signalling, Atomic-SDN takes the idea of SDN centralisation further: allowing a controller to define *what type of control service will happen next*, *when that service will happen*, and *how long it will take*.

[Q3]: *Are there solutions to reduce SDN overhead?* When considering the complexity of SDN control operations in a wireless mesh, the *one-to-all* communication pattern of SF is ideal for satisfying SDN *configuration* services from a central controller. Not only does this allow the

controller to configure or respond to multiple nodes in a single message, without having to communicate with each node in turn, but it means that the source routing approach championed in μSDN is no longer necessary: in a single flood, the controller can individually configure the flowtable of all nodes along a desired path. This returns some of the fine-grained configurability that is lost in other SDN approaches, whilst performing this operating in a fraction of the time.

[Q4]: *Can SDN scale in a low-power wireless mesh?* While the evaluation of Atomic-SDN in Section 5.4 doesn't touch on its scalability in large mesh networks (considered by this thesis to be multi-hop networks greater than 100 nodes), the SF-control plane made possible by Atomic-SDN's novel architecture means it scales far more readily than the approaches taken in current low-power wireless SDN solutions, including the lightweight μSDN architecture introduced in Chapters 3 and 4.

This is achieved in two ways. Firstly, the underlying SF protocols support communication at theoretical minimal bounds of latency, and the *one-to-all* traffic pattern in SF means messages are broadcast to all participating nodes. This allows Atomic-SDN to facilitate SDN services to all nodes in the shortest possible time.

Secondly the high-reliability of SF approaches means a single flood provides PDR of up to 99.99% within a single flood [94]. Although Atomic-SDN consumes all network resources within a flood, there are almost no retransmissions, and time spent on any given control period is minimised. As Figure 5.14a in Section 5.4 demonstrates, control services for the whole network can be delivered in under a second, compared to tens of seconds, or even minutes depending on the chosen MAC layer.

5.6.2 Further Consideration

This chapter has shown how the novel approach taken by Atomic-SDN allows it to rapidly and reliably reconfigure all nodes across the mesh in times that are orders-of-magnitude faster than current low-power wireless SDN approaches, including the work presented in Chapters 3 and 4. Furthermore, it is able to survive harsh wireless conditions that would normally cripple the SDN control plane. Not only does this solution therefore address many of the challenges that have made research into low-power wireless SDN solutions so complex but, importantly, it opens a number of intriguing new avenues for research.

Energy Efficiency: Though this chapter repeatedly emphasises the impressive reliability of SF protocols under external interference, this can only be delivered through considerable temporal and frequency redundancy. This increases the radio-on time in any given flooding round, and reduces the energy efficiency of the network as all nodes listen for transmissions. Furthermore, as with slotframes in IEEE 802.15.4 TSCH [83], the Layer-3 latency of communications are bounded by the period (i.e. minimum latency is capped by a node's next opportunity to transmit). To reduce delay, the SF protocol needs to reduce the epoch period, which in turn reduces the opportunity to allow nodes to sleep and save energy. Consideration therefore needs to be given on how best to

balance these needs, as well as adopting new techniques such as the *many-to-many* approach in Mixer [98], which can reduce the SF communication time.

Scalability in a Massive Mesh: While Atomic-SDN undoubtedly provides scalability in a local mesh of less than 100 nodes, there are legitimate questions as to how well this approach can scale for massive IIoT mesh networks of 1000 nodes or more. As alluded to in Chapter 2, though some recent works have explored the scalability of concurrent transmissions in a one-hop radius, there are few examples of a comprehensive examination of the scalability of SF approaches in a multi-hop scenario. The results presented in this chapter, and successive years of the IEEE EWSN Dependability Competition, have shown that SF protocols work well in mesh networks of 50→100 nodes. Yet the reader should be aware of some key limitations in these evaluations. Firstly, these network sizes typically don't exceed ~7 hops and there is little available research exploring how to maintain synchronisation at higher hop counts. Secondly, as most works examine CT at 2.4GHz, it is not yet well-understood how SF protocols would perform on the sub-GHz bands common in extremely large IIoT networks, where transmissions over large distances may start to affect phase offsets at the receiver. While there is scope to explore this problem through stochastic geometry and/or simulation, there are no readily available 1000+ node testbeds for real-world experimentation. As such, this is an area of research that would need considerable resources to properly evaluate.

Other Physical Layers: With recent research successfully demonstrating SF working on Bluetooth Low Energy (BLE) [78, 95], and Ultra Wide Band (UWB) [96] it is likely that much of the research going-forward will explore the application of SF in combination with various other physical layers. This presents an opportunity to utilise Atomic-SDN as an SF framework to dynamically schedule floods across various technologies and use an Atomic-SDN node as a universal controller for multiple client networks.

Industrial Wireless Control: The dynamic configurability of the underlying SF control architecture in Atomic-SDN means it can rapidly respond to changing control requirements. There is therefore considerable potential for extending Atomic-SDN for industrial wireless control. Recent research has shown that SF protocols are capable of providing the latency and reliability guarantees necessary to control (some) time-critical systems [139, 140], and the dynamic scheduling provided by Atomic-SDN would allow such systems to more readily adapt and reconfigure to changing application requirements.

Integration with 6TiSCH: This chapter specifically focuses on how SF protocols can provide a reliable, low-latency SDN control plane, and while Section 5.3 defines a protocol capable of manipulating the SDN flowtable, a promising area of research is how this then might be integrated with 6TiSCH in order to provide dynamic configuration of the TSCH slotframe. Although 6TiSCH can schedule network resources and help provide scalability to support extremely large mesh networks for IIoT, channel hopping and duty-cycle restrictions mean it can take a considerable amount of time to reconfigure a TSCH schedule in response to changing service requirements.

In a 400 node real-world network operated by the industrial sponsor of this PhD, it can take 30 minutes to propagate schedule changes across the network, while Atomic-SDN has the potential to reduce this to seconds.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

This PhD has explored the application of Software Defined Networking (SDN) concepts within the scope of Industrial Internet of Things (IIoT). Yet while SDN research has successfully been applied in data centre and cloud networks [15–18], where it is supported by low-latency wired connections and powerful hardware switches, the centralised control architecture at the heart of SDN introduces complex and varied signalling that is ill-suited to the multi-hop mesh networks common in industrial sensing and control scenarios. To this end, the publications supporting this thesis have focused on overcoming this challenge; focusing on the IEEE 802.15.4-2015 [1] standard, exploring novel solutions to address the additional overhead and providing reliable, timely communications for programmable SDN control solutions for low-power wireless networks.

6.1 Chapter Summary

Chapter 1 introduced the challenge of applying an architecture originally intended for high-performance campus networks to constrained wireless mesh networks, where low-power narrowband communications can be prone to interference over a shared spectrum, are subject to duty-cycle limitations, and multi-hop communications make the distribution of network state a highly complex problem. This chapter framed the challenge through five research questions and, throughout the manuscript, subsequent chapters have endeavoured to address these questions in their conclusions.

[Q1]: *What are the fundamental features of a minimal SDN solution?*

[Q2]: *How does SDN control signalling affect the low-power wireless mesh?*

[Q3]: *Are there solutions to reduce SDN overhead?*

[Q4]: *Can SDN scale in a low-power wireless mesh?*

[Q5]: *What advantages does SDN bring to a low-power wireless mesh?*

Chapter 2 - [Q1, Q2, Q3, Q4]: The background material covered in Chapter 2 consequently addressed recent research and standardisation efforts in SDN and IEEE 802.15.4, as well as relevant literature addressing SDN for low-power wireless networks. Additionally, outside of current low-power wireless standards, the concept of Synchronous Flooding (SF) was discussed as a means of providing low-latency and highly reliable communications across a mesh network.

Chapter 3 - [Q1, Q2, Q3, Q5]: The design and implementation μSDN was presented in Chapter 3. This low-overhead SDN stack has subsequently been made available as a publicly available repository for academic research purposes [R1], and has recently been used to support research into energy-aware SDN solutions for low-power IoT networks [C4]. The μSDN architecture leverages RPL [24] interoperability to provide a distributed solution to SDN controller discovery and maintenance, a weakness in previous literature, and implements a technique to inject RPL source routing headers directly from the SDN flowtable in order to minimise the number of messages exchanged between the mesh and SDN controller. Furthermore, this chapter introduced μSDN-Atom, a layered SDN controller architecture capable of running on constrained hardware with just a few KB of memory.

Chapter 4 - [Q1, Q2, Q4]: μSDN is extended in Chapter 4 to incorporate efforts from the IETF 6TiSCH WG [2]. Taking the concept of 6TiSCH tracks, in which deterministic Layer-2 forwarding paths are scheduled across the mesh by reserving IEEE 802.15.4e-2012 Time Scheduled Channel Hopping (TSCH) [83] slotframe and buffer resources, this chapter demonstrates how tracks effectively isolate the SDN control plane, and mitigate the cost of the additional overhead by eliminating contention with other network traffic. Yet although these results highlight that the contention issues introduced by SDN overhead can be alleviated with network slicing, each track involves monopolising a considerable portion of network resources. Although this significant use of resources could perhaps be mitigated with further research into the concepts of complex and shared tracks, as defined within the 6TiSCH standard, in this author's view the point-to-point traffic patterns supported by 6TiSCH tracks are not suited to providing a scalable solution to the SDN control plane - particularly when considering of extremely large IIoT networks.

Chapter 5 - [Q2, Q3, Q4]: However, while the first two technical chapters focused on how SDN can work within the existing IEEE 802.15.4-2015 standardisation efforts, a key contribution of this thesis has involved research into how Synchronous Flooding (SF) techniques can provide low-latency, high-reliability communications in an unreliable narrowband environment. Not only

does SF inherently broadcast, allowing a controller to rapidly configure the mesh in a single flood, but this time-synchronised technique temporally slices the network allowing other network processes to operate without contending with SDN control signalling. To this end, Chapter 5 presented Atomic-SDN, a novel architecture capable of scheduling multiple SF protocols in order to support the core SDN control services identified in the initial chapters. As evidenced by its all-round performance in the 2019 IEEE EWSN Dependability Competition (placing 2^{nd} in both categories) the Atomic-SDN approach successfully adapts to a variety of complex traffic patterns in highly dynamic environments. While there are still challenges concerning the scalability of SF over large distances, Atomic-SDN can perform SDN control signalling in a fraction of the time of current SDN solutions for low-power wireless. As such, it provides a platform to help support programmable control architectures for IIoT. Not only for industrial wireless control in factory-scale networks, but also providing a foundation for further research into rapid and reliable configuration of extremely large-scale Advanced Metering Infrastructure (AMI) mesh networks.

6.2 Recommendations

Based on the results and findings in this thesis (supported by the publications listed in Chapter 1) the following recommendations are made to researchers and engineers whose research addresses SDN concepts, or centralised control solutions in general, for low-power wireless networks.

- **Recommendation 1:** *(Short-Term) Use SF to provide accurate time-synchronisation and centralised SDN control of the 6TiSCH slotframe.*

  In comparison to the tight time-synchronisation in SF (where nodes must transmit within 0.5μs of each other, depending on the physical layer), 6TiSCH provides generous guards and tolerates some drifting before nodes disassociate [141, 142]. By encapsulating SF within one or more TSCH slots (depending on the desired N_{Tx}), the inherent time synchronisation and broadcast communications of SF could replace 6TiSCH Enhanced Beacons (EBs). This would achieve two things: firstly, as SF ensures tighter time synchronisation and a smaller drift, this would allow a reduction in guard times and thus shorter TSCH slots, freeing up capacity within the slotframe, and allowing end-to-end latencies nearer to the maximum throughput of the physical layer; secondly, the low-latency SF broadcast communications would support dynamic centralised scheduling of the TSCH slotframe, allowing complete reconfiguration of the slotframe in under a second, as opposed to minutes or tens of minutes in current 6TiSCH deployments. This idea, of being able to rapidly reconfigure the network in response to changing application requirements, is central to the SDN concept, and is crucial if constrained low-power networks, with limited available resources, are to support multi-application and multi-tenant networks for IoT.

- **Recommendation 2:** *(Medium-Term) Extend SDN architecture to address the multiple PHY layers available for low-power wireless IoT.*

  Chapter 2 introduced the IEEE 802.15.4-2015 standard and the myriad of various lower layer options available. While this thesis has focused on IEEE 802.15.4 OQPSK-DSSS at 2.4GHz, it is this author's view that the SDN concept must be extended to incorporate multiple solutions for IoT communications: not just in IEEE 802.15.4, but also incorporating other emerging low-power wireless technologies. Particularly when considering recent Bluetooth standardisation, which provides higher rate communications of up to 2MBPS, multi-PHY solutions are attracting considerable research interest, and future research may investigate how appropriate physical layers can be chosen to balance latency, reliability, energy, and distance benefits. Given programmable reconfigurability is at the heart of SDN, this presents an opportunity to use the architecture to handle multi-PHY communications on a per-flow basis.

- **Recommendation 3:** *(Long-Term) Explore how SDN can be scaled to help support massive mesh networks for IIoT.*

  Further to Recommendations 1 and 2, while efforts from the 6TiSCH WG have supported the scaling of low-power wireless through scheduling of co-located transmission over orthogonal frequencies, there is no one protocol or technique capable of supporting all massive mesh scenarios for IIoT. It is here that an SDN approach to mesh communications looks most promising. By their nature, extremely large mesh networks will likely operate across wide areas, where the radio environment in one area can wildly differ from the environment in another. The network may include both sparse and dense topologies, be tens of hops across, and sub-domains may be locally subject to external interference from nearby devices. It is this author's view that in order to support such diverse networks, the mesh needs to be managed on a case-by-case basis, where SDN supports the programmable configuration of local clusters based on current application and environmental needs. In addition to the recommendations above (using SF to support fast configuration and adopting multiple physical layers), this could also involve configuring specific sections of the network to adopt different MAC protocols based on current needs. However, while Chapters 3 and 4 showed that SDN signalling overhead could be reduced to levels that can be supported by RPL-based tree topologies with a limited number of nodes, formal evaluations into the greater scalability of this solution are lacking. In light of the considerable effort spent on the development of the μSDN networking stack, the public release of the codebase presents an opportunity for research around scalability, and how to properly evaluate and benchmark low-power wireless SDN mesh solutions with 100s, 1000s, or even 10,000s nodes.

6.3 Further Work

In addition to the above recommendations, the following specific items are suggested as future work within standardisation activities, as well as novel research areas examining how SDN can support next generation IIoT networks.

- **Explore hierarchical, ad-hoc, and east-west distributed control architectures to support SDN scalability in IoT mesh.**

  Current literature reviewed in Chapter 2 highlighted how distributing control intelligence across the mesh could help to further reduce SDN control signalling and in turn support scalability. While these works introduce hierarchical control architectures, allowing local controllers to manage a small sub-domain, there has yet to be any extensive research fully evaluating how, where, and to what degree SDN controllers should be distributed within the mesh in order to support massive scalability. Indeed, recent works such as Whisper [116] suggest that not all nodes even need to be fully capable SDN nodes, and that they can be placed strategically within the network in order to manipulate their local area. This author believes there is considerable scope to explore the benefits and limitations of distributing SDN controller intelligence within the mesh, either through the embedded μSDN-Atom controller or some other means, which could possibly be combined with distributed approaches such as Whisper to develop novel control topologies and hierarchies.

- **Align current SDN research with 6TiSCH track standardisation, incorporating tracks within the wider low-power wireless SDN architecture.**

  As most of the activity within 6TiSCH has focused on layer-3 routing, it is thought 6TiSCH will either need to be re-charted in order to fully define how tracks are installed across the TSCH slotframe, or that they could be included as part of a possible new IETF WG on Reliable and Available Wireless (RAW) [127]. This provides an excellent opportunity to help further extend SDN concepts within future standardisation efforts for low-power wireless, and open new research areas concerning complex tracks and how they can support end-to-end communications over heterogeneous MAC and PHY layers.

- **Explore Network Function Virtualisation (NFV), in the context of low-power wireless IoT, through the development μSDN control layer applications.**

  The focus of this thesis has been on how to support the SDN control plane in an unreliable multi-hop mesh environment. The μSDN architecture covered in Chapters 3 and 4 proposed and implemented mechanisms to achieve this, as well as incorporating selected concepts from literature. However, key features have yet to be evaluated. In particular, the μSDN flowtable supports in-network packet processing, such as data aggregation, and the ability

to manipulate flows based on the local network state. These could be used to explore NFV to help with network load balancing, avoid interference, and dynamically implement new protocols. Yet there is still no consensus on how best to do this, what IoT use-cases are best served by a programmable SDN architecture, and what sort of applications should be hosted at the controller when considering the constraints of the underlying mesh.

- **Explore issues of scalability in Concurrent Transmissions (CT), with the aim of using CT as an SDN control layer for extremely large wireless mesh networks.**

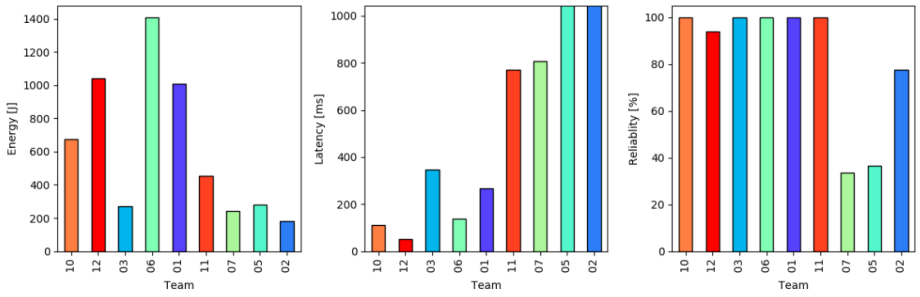
  While this thesis supports CT and SF as a solution for fast and reliable control in mesh networks, there is currently limited research into the properties behind in low-power wireless (although the author is aware of a number of efforts to address this), and what research there is can sometimes present conflicting results. In particular, Chapter 2 highlighted how scalability can be an issue in SF-based communications. Firstly, some works find that in a single hop scenario increasing the number of concurrently transmitting nodes will eventually start to reduce reliability, leading to questions over the technique's usefulness in particularly dense topologies. Secondly, most of the research has been performed at 2.4GHz at a few tens of meters and there are doubts as to how well the current SF synchronisation method (estimating the time of transmission) works at sub-GHz bands, given propagation delays and multipath over long distances. Finally, current research has been limited to networks of under a hundred nodes where the TelosB hardware, commonly employed in previous literature, has struggled with drifting at larger hop counts. Although some of these drift issues may be improved as the community migrates to modern hardware platforms, in an extremely large AMI network where hop distances can be tens of nodes there needs to be consideration as to how synchronisation can be maintained at the peripheries of the network, away from the timesync.

APPENDIX A

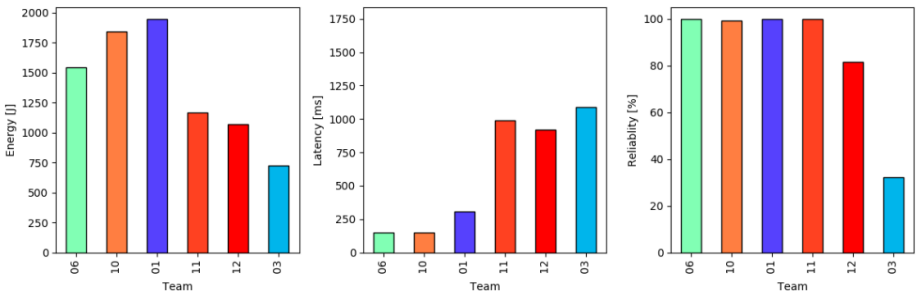
IEEE EWSN 2019 DEPENDABILITY COMPETITION

In this appendix results from the EWSN 2019 Dependability Competition are provided in detail[1]. Figures cover both the *data collection* and *data dissemination* categories and show energy, latency, and reliability across each jamming level (none, strong, strong, and dynamic) for all three message lengths (32b, 32b, 64b). Results are a median average of both of layouts, and all three traffic loads (periodic 5s/30s and aperiodic). Team *06* denotes the ASDS solution covered in Section 5.5.

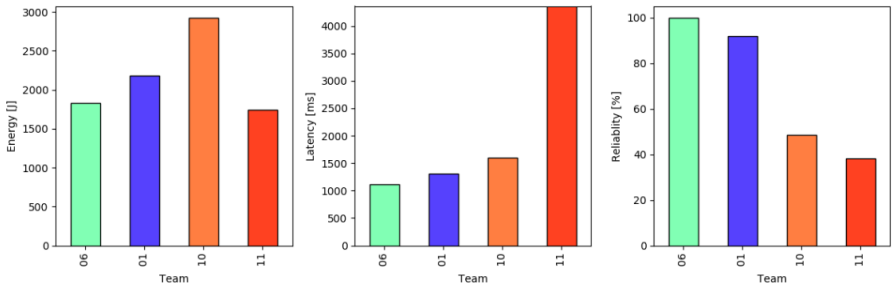
[1]This author has obtained permission from the competition organisers to reproduce these results in this thesis.



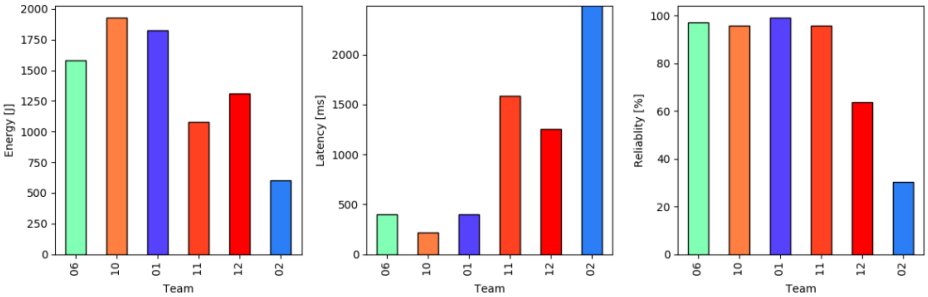
(a) No jamming.



(b) Mild jamming.

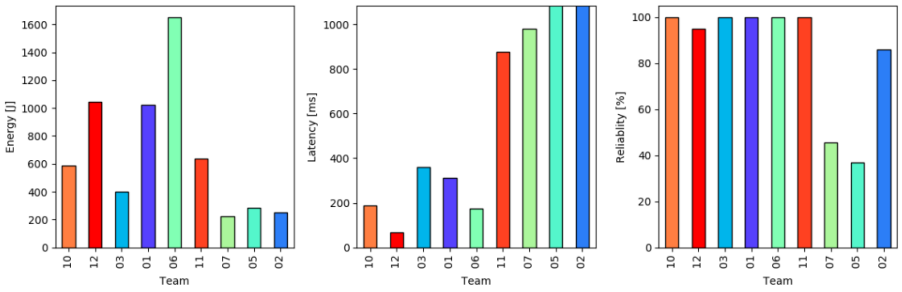


(c) Strong jamming.

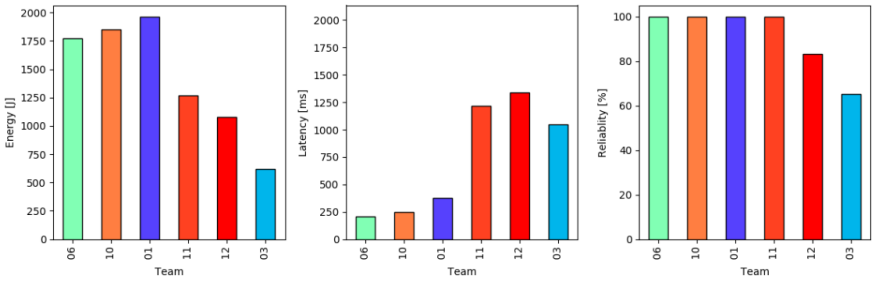


(d) Dynamic jamming.

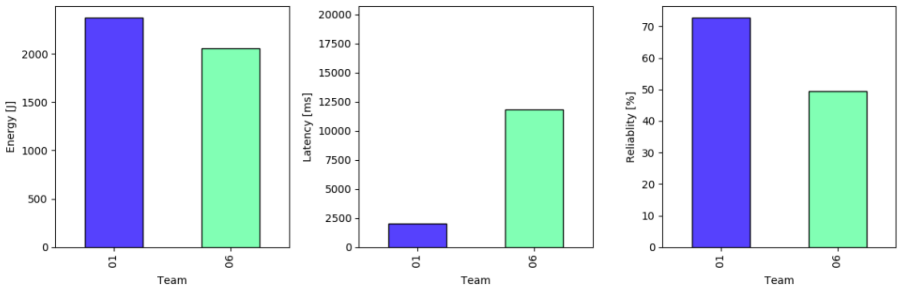
Figure A.1: Median results from data collection category with 8b message length.



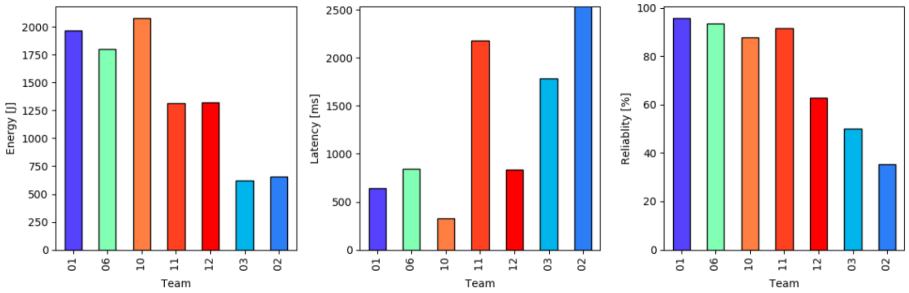
(a) No jamming.



(b) Mild jamming.

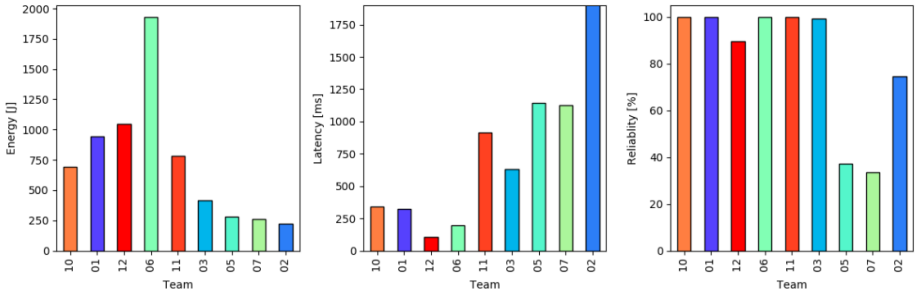


(c) Strong jamming.

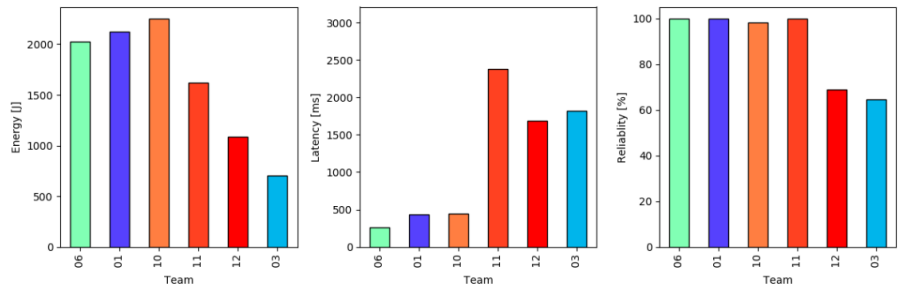


(d) Dynamic jamming.

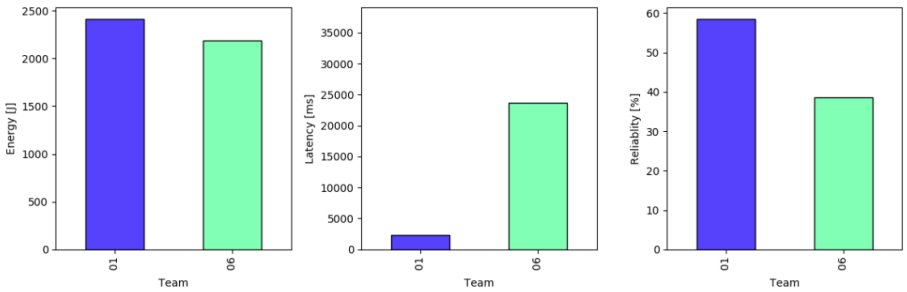
Figure A.2: Median results from data collection category with 32b message length.



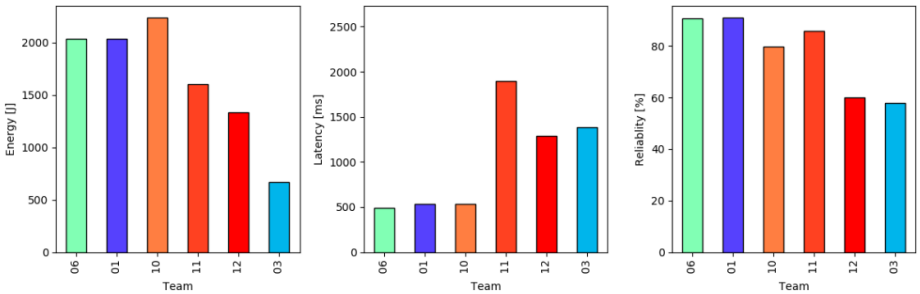
(a) No jamming.



(b) Mild jamming.

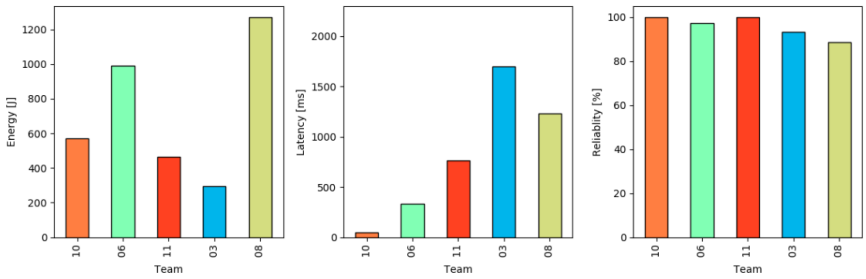


(c) Strong jamming.

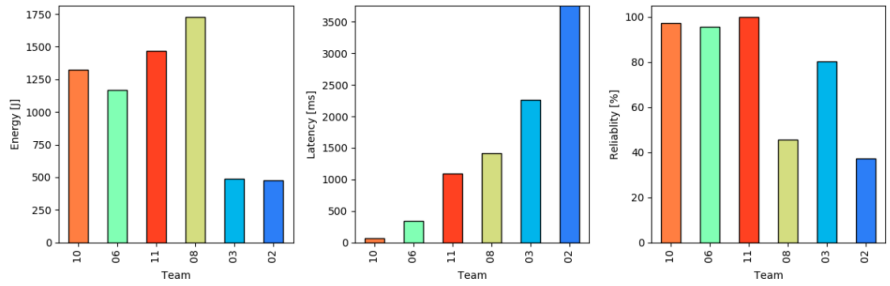


(d) Dynamic jamming.

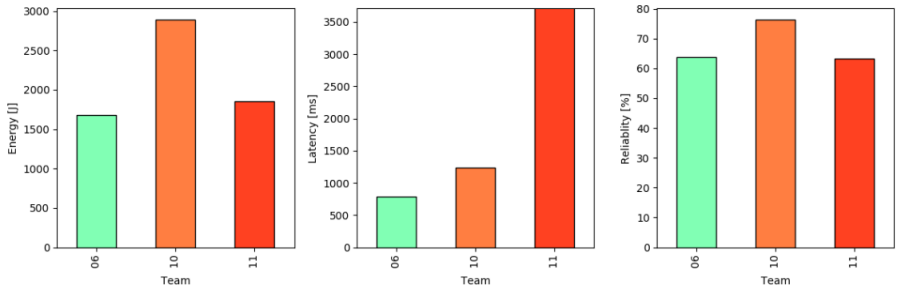
Figure A.3: Median results from data collection category with 64b message length.



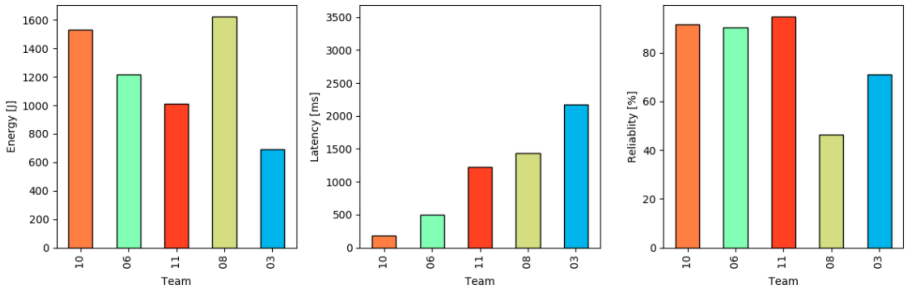
(a) No jamming.



(b) Mild jamming.

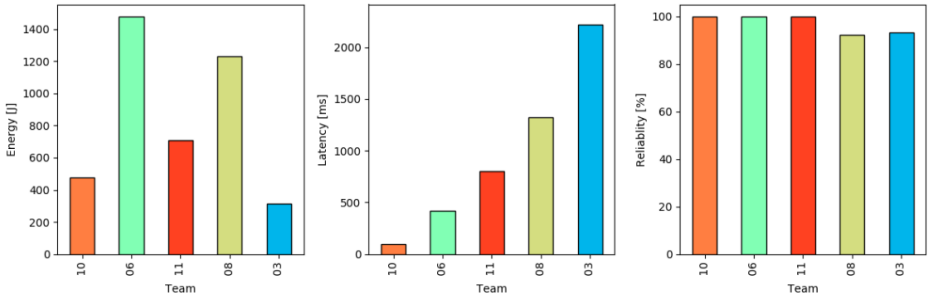


(c) Strong jamming.

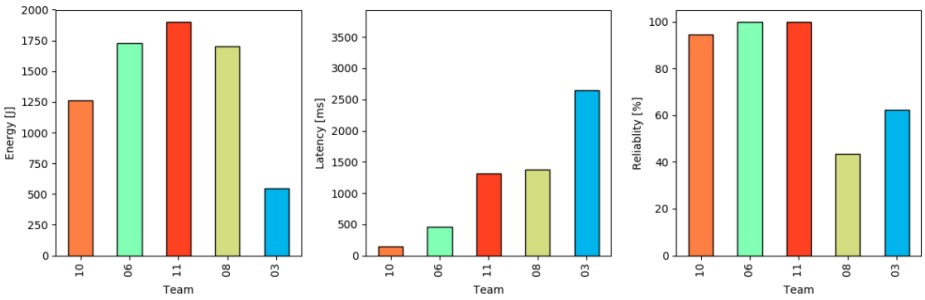


(d) Dynamic jamming.

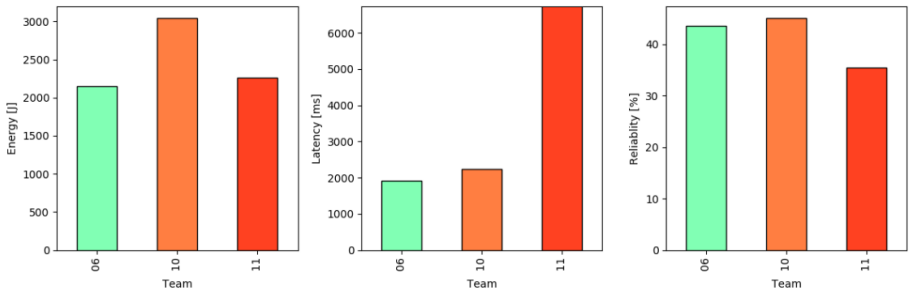
Figure A.4: Median results from data dissemination category with 8b message length.



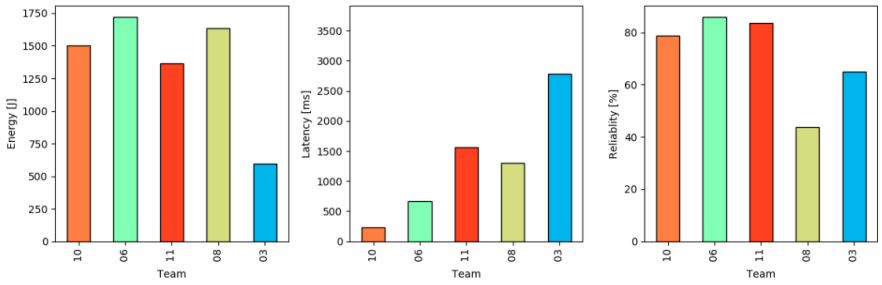
(a) No jamming.



(b) Mild jamming.

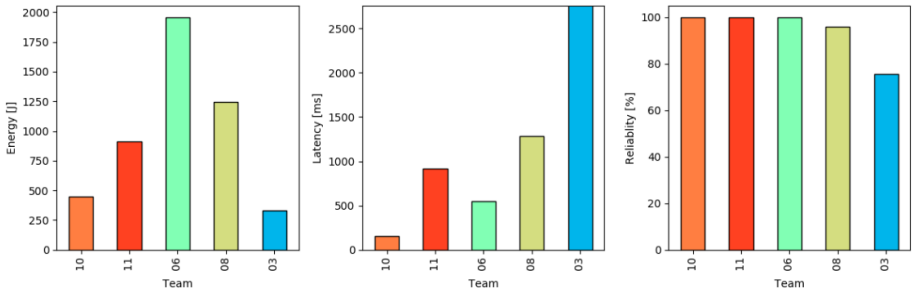


(c) Strong jamming.

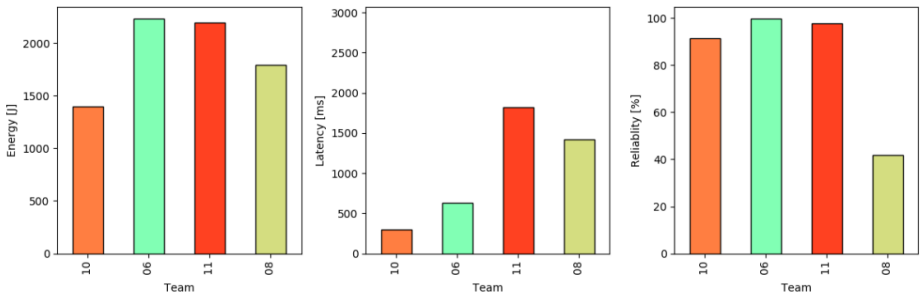


(d) Dynamic jamming.

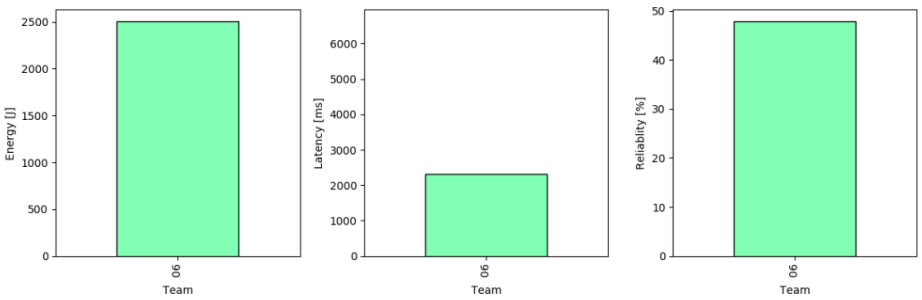
Figure A.5: Median results from data dissemination category with 32b message length.



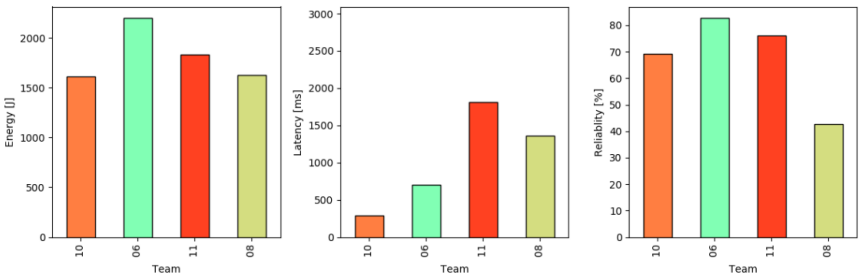
(a) No jamming.



(b) Mild jamming.



(c) Strong jamming.



(d) Dynamic jamming.

Figure A.6: Median results from data dissemination category with 64b message length.

APPENDIX B

US PATENT APPLICATION NO. 16/176659

In this appendix, patent application no. 16/176659 is included: *"A Controller for, and a Method of Processing Data Over, a Low-Power, Wireless Software Defined Networking, SDN Architecture"*, filed in the Unites States on the 31st of October 2018.

A CONTROLLER FOR, AND A METHOD OF PROCESSING DATA OVER, A LOW-POWER, WIRELESS SOFTWARE DEFINED NETWORKING, SDN, ARCHITECTURE

FIELD

Embodiments described herein relate generally to software defined networking and in particular to configuring a software defined network.

BACKGROUND

Software defined networking (SDN) enables central programming of different services across a network, including a plurality of interconnected nodes. There have been some SDN architectures which have been implemented for IEE 802.15.4 low power wireless networks. The shared nature of the underlying wireless medium, a need to transmit data over multiple hops, and stringent resource constraints pose significant challenges not present in a conventional wired network.

SDN requires frequent back and forth communication between a central controller and the network nodes. This flow of traffic requires a variety of different communication types, including one-to-one, one-to-many, and many-to-one communication. Current communication protocols, e.g. Routing Protocol for Low-power or Lossy (RPL) networks, are not optimized to deal with more than one communication type, or traffic pattern. For instance, RPL is typically used to implement converge-cast, or many-to-one, traffic. Whilst it is possible to use an extension to extend the available communication types to include one-to-one and one-to-many communication, for example, such extensions incur significant overhead, especially in terms of additional storage required at the network nodes.

For the avoidance of doubt, many-to-one communication, also known as “collection”, may be used to allow the controller to gather information from the nodes of the network for understanding the current state and performance of the network. In addition, one-to-one communication, also known as “reaction”, may be used to allow nodes to solicit or alert the controller for instructions on how to react to new input or events. Furthermore, one-to-many communication, also known as “configuration”, may be used to allow the controller to provide instructions to all nodes within the mesh, or subset thereof.

It is an object of the present disclosure to improve on the prior art.

Arrangements of the present invention will be understood and appreciated more fully from the following detailed description, made by way of example only and taken in conjunction with drawings in which:

Fig. 1 shows a block diagram of a controller according to an embodiment;

Fig. 2 shows a time line of network operation using the controller from Fig. 1;

Fig. 3 shows a set of opportunities denoting communication types to configure the network using the controller from Fig. 1;

Fig. 4 shows a data format of the phases;

Fig. 5 shows a set of one-to-all phase types;

Fig. 6 shows a set of many-to-one phase types;

Fig. 7 shows a set of one-to-many phase types;

Fig. 8 shows a stop phase-pair;

Fig. 9 shows a schedule of opportunities;

Fig. 10 shows a schedule of the react opportunity from Fig. 9;

Fig. 11 shows a timing structure of a phase;

Fig. 12 shows a flow chart of a feedback loop configured through the network;

Fig. 13 shows a flow chart of an opportunity;

Fig. 14 shows a latency profile at each hop for the controller from Fig. 1 compared to two conventional controllers;

Fig. 15 shows the time to join nodes in a network using the controller from Fig. 1 compared to two conventional controllers;

Fig. 16 shows the end-to-end PDR and radio duty cycle for the controller from Fig. 1 compared to two conventional controllers;

Fig. 17 shows the scalability of the opportunity types using the controller from Fig. 1; and

Fig 18 shows the controller from Fig. 1 in communication with a network.

SUMMARY

According to an embodiment, there is provided a controller for a low-power, wireless, software defined networking, SDN, architecture, comprising a processor and a memory storing instructions for execution by the processor, the instructions, when executed by the processor causing the controller to implement: a control layer arranged to detect a communication type for processing an application over the network, and translate the communication type into a plurality of phases chosen from a pre-defined set of phases to configure the nodes of the network to respond to the application according to the communication type. Translating the communication type into a plurality of pre-defined phases allows for the network to be configured according to the application to be transmitted over the network, without incurring undesirable overhead.

The communication type may be selected from a list including: one-to-one communication; one-to-many communication; and many-to-one communication.

The controller may configure the nodes to respond to the application by selecting one mode from a list including: source mode; destination mode; and forwarder mode.

The phases may be scheduled in a time synchronous manner. In this way, latency is time bound and a minimum and maximum delay guarantee may be provided to SDN services and applications.

Each phase may include a generic flood primitive to configure each node of the network.

Each phase may include pre and post processing logic, the pre-processing logic may be arranged to modify the generic flood primitive based on the current performance of the network, and the post-processing logic may be arranged to detect a performance of the network in responding to the generic flood primitive. Pre and post processing provide a feedback function to allow the network to adapt to changing circumstances before and after each flooding phase.

Each phase may include a guard time period. The guard time period allows for drift and processing in other nodes.

Each phase may include an offset time period. In this way, the start and end point of a data packet are known.

The controller may be further configured to divide a data processing epoch into a control plane and a subsequent data plane, wherein the controller may be configured to transmit the phases to the network in the control plane, and may be configured to transmit the application data to the network in the data plane.

According to another embodiment, there is provided a low-power, wireless, software defined network, SDN, comprising: a plurality of nodes; and the controller described above arranged to configure the nodes of the network to process the application over the network.

According to another embodiment, there is provided a method of processing data over a low-power, wireless, software defined networking, SDN, architecture, comprising: detecting a communication type for processing an application over the network, and translating the communication type into a plurality of phases chosen from a pre-defined set of phases to configure the nodes of the network to respond to the application according to the communication type.

The communication type may be selected from a list including: one-to-one communication; one-to-many communication; and many-to-one communication.

The nodes responding to the application may include acting in one mode selected from a list including: source mode; destination mode; and forwarder mode.

The method may further comprise scheduling the phases in a time synchronous manner.

Each phase may include a generic flood primitive to configure each node of the network.

The method may further comprise detecting a performance of the network in responding to the generic floor primitive and modifying a subsequent flooding primitive based on the performance of the network.

The method may further comprise scheduling the phases to include a guard time period.

The method may further comprise scheduling the phases to include an offset time period.

The method may further comprise: dividing a data processing epoch into a control plane and a subsequent data plane, transmitting the phases to the network in the control plane, and transmitting the application data to the network in the data plane.

The method described above may be a computer-implemented method.

According to another embodiment, there is provided a non-transitory memory storing computer program instructions for execution by a processor, the instructions, when executed by the processor, causing the processor to perform the method describe above.

DETAILED DESCRIPTION

With reference to Fig. 1, the present embodiment relates to a controller 10 for a low-power, wireless, software defined networking (SDN) architecture. The controller 10 includes an SDN control layer 12, an Atomic-SDN layer 14, and a synchronous flooding layer 16.

The controller 10 is arranged to detect a communication type required by an application being processed over the network. The application may require one-to-one communication, many-to-one communication, or one-to-many communication, for example. The SDN control layer 10 is arranged to detect the required communication type and communicate the communication type intent to the Atomic-SDN 14.

The Atomic-SDN 14 includes an SDN control application programming interface (API) 18, an opportunity scheduler 20, and an abstract protocol builder 22.

The SDN control API 18 is arranged to translate the communication type intent from the SDN control layer 10, to an opportunity. The term “opportunity” is used to mean a period in which the controller initiates control across the network. The type of opportunity is chosen by the controller prior to a flooding period. An opportunity

includes a set of logic arranged to configure the nodes of the network to transfer data according to one of the abovementioned communication types. The logic is constructed through one or more phase types (described below), along with pre and post processing logic.

A "phase" is a building block of the opportunities. Higher level functions are able to be created through chaining multiple phases into a series of logic decisions. Each phase is a self-contained unit consisting of a flood primitive, configured with N transmissions and duration Tdur, combined with an associated data structure and certain functions which define phase behaviour based on the current node role. The functions include pre and post processing logic, guard to allow for drift and time for processing to occur in other nodes, and offset from an initial phase reference.

By defining these functions, phases can be configured to perform a specific, self-contained role, whilst propagating the associated phase packet types (see Fig. 12). Multiple phases can then be chained together in order to build up higher level processes (see Fig. 3), allowing full protocols to be implemented through the combination of a number of simple blocks.

A "flood primitive" is a generic function defined as a flood configured with N number of transmissions and duration Tdur. If a node is able to successfully complete all N transmissions, it will exit the flood process, otherwise it will exit at Tdur. Flood primitives are currently implemented as a one-to-all synchronous flood, however, any lower synchronous flooding layer could conceivably be used.

The opportunity scheduler 20 is arranged to receive the proposed opportunities and schedule the opportunities to a time when they are required in order to process the application. For instance, different stages of the application may require different communication types, and so the opportunity scheduler arranges the opportunities in a time schedule. The abstract protocol builder 22 then creates a protocol according to the opportunity to realise the one-to-one, one-to-many, and many-to-one configuration across the network.

The abstract protocol builder 22 includes: opportunity 24 and phase 26 pre-processors, opportunity 28 and phase 30 post-processors, the generic flood primitive 32 for the phase, and guard 34 and offset 36 processors. The abstract protocol builder 22 also includes a phase scheduler 38.

Functionally, the abstract protocol builder is arranged to determine the phases required to realise the opportunity from the opportunity scheduler and construct a

protocol from the selected phases. The opportunity and phase pre-processors 24, 26, are arranged to adapt the flood primitive of the selected phase according to the current network state. The phase and opportunity post-processors 28, 30, are arranged to observe a state of the network resulting from applying the opportunity to the network. The guard and offset processors are arranged to apply the phase guard and offset times to the opportunity. The phase scheduler 38 is arranged to schedule the phases according to the opportunity.

The synchronous flooding layer 16 is arranged to flood the network in accordance with the opportunities.

Fig. 2 shows a time line of data transmitted over the network. There are three epochs shown in full. An epoch is defined as a period of time between regularly scheduled SDN control opportunities, with periodicity Ti, where a trade-off is considered when setting the epoch length, and consequently the frequency of SDN opportunities. As synchronous flooding periods according to embodiments inherently block other processes, a longer epoch allows a greater amount of time to be devoted to normal network operation; whether that is application processes or to allow nodes to sleep and therefore conserve energy.

With reference to Fig. 2, an epoch 40 includes a period of SDN operation 42, and a period of normal operation 44. During the SDN operation period 42, the SDN controller is arranged to configure the nodes of the network to operate according to one of the communication types. During the normal operation, application data is processed over the network as configured in the selected communication protocol.

With reference to Fig. 3, there are three opportunities 48 a, b, c, used by the controller according to an embodiment. The three opportunities 48 a, b, c, include: “collect opportunity”; “react opportunity”; and “configure opportunity”. The collect opportunity corresponds to a many-to-one communication type. In other words, during a collect opportunity, the nodes of the network are configured such that all, or a subset, of the nodes are arranged to transfer data to one designated node. The react opportunity corresponds to a one-to-one communication type. In other words, during a react opportunity, the nodes of the network are configured such that one node is configured to transfer data to another designated node. The configure opportunity is a one-to-many communication type. In other words, during a configure opportunity, one of the nodes in the network is configured to transfer data, and a subset of other nodes are configured to receive the data.

With further reference to Fig. 3, the opportunities 48 a, b, c, include a set of phases 50. The phases 50 include data packets serving a specific function. The specific phases used for each opportunity are selected from a pre-determined set of phases. Each opportunity starts with an indication phase (IND). The indication phase (IND) is a one-to-many phase and is arranged to communicate the opportunity type, and duration of the control period, to the rest of the network.

Fig. 4 shows a message format of a phase 50, in this case an indication phase (IND). The opportunity type is communicated first in the Op Type field 1B, followed by data relating to the node ID flags in field 4B. The message size is described as 5 Bytes; 1 Byte for the opportunity type, and four Bytes for the node ID flags.

Figs. 5-8 show phases according to different communication types. Fig. 5 shows phases where one node communicates a data packet to all of the other nodes, i.e. one-to-all phase types. In particular, there are Indication (IND), Acknowledge (ACK), and Negative-Acknowledge (NACK) phases. The IND phase is used for time synchronization of the network, and indicating the type of opportunity that the network will run next (i.e. collect/configure/react). Nodes can then make their own decision if they will participate as a source, a destination, or a forwarder. The data packet includes an opportunity type description (op type) and a bit array indicating which nodes are allowed to participate in this opportunity (N.B. non-participating nodes are still forwarders). The bit array indicating the nodes is labelled as “Node ID Flags”. The ACK phase is an acknowledgement from the controller to the network. The NACK phase is used as the final phase of an empty collect opportunity. The ACK and NACK phases each include a 3 bit description (although the size of the packet is not limited to 3 bits) of the phase type followed by a similar “Node ID Flags” as included in the IND phase.

Fig. 6 shows phases where many nodes transfer information to a single node, i.e. many-to-one communication. In particular, the many-to-one phase types include a solicit phase, and a report phase. The solicit phase is used in the react opportunity. The solicit phase may allow nodes to request information from the controller. The data structure of the solicit phase includes a type of phase (TYP) followed by a source of the packet which this node is soliciting instructions about (SRC), followed by a destination of the packet which this node is soliciting instructions about (DST), and the value of the packet which this node is soliciting instructions about (VAL).

The report phase is used in the collect opportunity. Report may inform the controller about the state of the node. The repose phase includes a type of phase (TYP) followed by an identifier, ID, of the node (Node ID), then N Bytes of metric data, where type and length of data required by the controller has been pre-configured on association with the network (DATA). Fig. 7 shows phases where many nodes transfer information in a one-to-many type communication. In particular, there is a set phase. The set phase allows the controller to send configuration data to the node. The data structure of the set phase includes a type of phase (TYP) followed by a bit array of nodes which should receive this packet (Node ID Flags). In parallel to those two data packets, there is a field which donates the index on which to perform SDN matching for regular packets labelled “Match IDX”, a field which donates the type of SDN matching operation (Equals/Less-Than/Greater-Than…) to perform on regular data packets which match this rule, a field which donates the byte value to match when performing SDN matching for regular data packets labelled “Match VAL”, a field which donates the type of SDN action operation (Forward/Drop/Modify/Aggregate…) to perform on regular packets which match this rule labelled “Action Op”, and a field which donates a byte value which can be used in conjunction with the SDN action operation (e.g. Forward to X/ Drop from Y…) which is labelled “Action VAL”.

It is noted that the phases and the opportunities are independently classed in one of the communication types: one-to-many, many-to-one, or one-to-one. However, the type of opportunity may require phases associated with a different communication type. For instance, in order to facilitate many-to-one communication, the opportunity is broken down into multiple phases, with each distinct phase performing a different communication type (in the case of the collect opportunity, the ACK and IND phases are used, which phases are categorised as one-to-all phases). The combination of these phases, and their consecutive traffic patterns, allows the network to perform the reliable many-to-one communication as seen in the collect and react opportunities.

With reference to Fig. 8, there is also provided a stop phase-pair. The stop phase-pair includes an empty phase and an acknowledge phase. The empty phase is a many-to-one phase, and the acknowledge phase is a one-to-all phase.

With reference to Fig. 9, the phases for each opportunity (see Fig. 3) are scheduled during the SDN operation phase over time Δxop, followed by a period of normal operation, where application data is transmitted over the network. Next, the phases of the subsequent opportunity are scheduled during the next SDN operation

phase. The phases can be seen as distinct, time synchronous, data packets transmitted to the network.

With reference to Fig. 10, the phases are bounded by a flood time and are separated according to a time schedule. In this way, the time for transmitting the opportunity to the network is time bounded resulting in time bounded latency. Since the phase structure of each SDN opportunity is known, the controller 10 can provide minimum and maximum delay guarantees (on successful control operations) to SDN services and applications.

With reference to Fig. 11, the SDN opportunity is scheduled to provide for a pre-processing phase, a guard phase, then a synchronous flooding period, and a post processing phase.

With reference to Fig. 12, the pre and post-processing features are best described with reference to the operational flow of the controller.

In step 100, a network application is called which is to be transferred over the network. The application requires a specific routing 102 between nodes, a particular quality of service 104, and is associated with specific security requirements 106. The intent of the application is determined from the network application at step 108, and the application intent is compiled at step 110.

At step 112, the SDN opportunity required to realise the application intent 110 is determined and the SDN opportunity is scheduled. After checking the synchronous flooding state at step 114, the synchronous flooding layer is configured to flood the network with the flooding primitives of the phases of the opportunity at step 116.

At step 118, the SDN opportunity for flooding to the network is indicated. In particular, the network will be flooded according to the collect opportunity 122, the react opportunity 124, or and configure opportunity 126. In this way, the network will be configured according to the chosen opportunity so that subsequent data transfer required by the application can be carried out in an optimised manner.

At step 120, the SDN opportunity is pre-processed.

At step 128, SDN opportunity post-processing occurs, where the state of the network is monitored to see how the network responds to the opportunity.

At step 130, the network state is updated, and also the application view and the synchronous flooding state, are also updated in steps 132 and 134. In particular, the controller includes an application view data base 136, a network state database 138,

and a synchronous flooding state database 140 to maintain information relating to the application view, the network state, and the flooding state, respectively. These databases are updated in steps 130, 132, and 134. The application view database can be used to call information relating to routing on the network, quality of service available on the network, and security information of the network.

Phase scheduling for an opportunity is described with reference to the process flow chart in Fig. 13.

At step 200, the opportunity is pre-processed. Once processed, the phases required to realise the opportunity are selected at step 202 (see Fig. 3 for phases required per opportunity). At step 204, the phases are pre-processed according to the network state. The guard time, the offset, and the duration, are applied to the phases at steps 206, 208, and 210. Once the phases have been scheduled, the opportunity is flooded to the network at step 212.

At step 214, the phases are post processed to understand how the network responded to the phases. If the opportunity was completed in the duration, Tdur, the opportunity is post processed (see above). If the opportunity was not completed, the process reverts back to step 202.

The benefits of the controller 10 according to embodiments are best understood with reference to the test results shown in Figs. 14-16.

With reference to Fig. 14(a), the mean collect delay is shown as being lower above 3 hops compared to the uSDN-CSMA and uSDN-CONTIKIMAC protocols. With reference to Fig. 14(b), the mean react delay is shown as being lower above 1 hop compared to the uSDN-CSMA and uSDN-CONTIKIMAC protocols. With reference to Fig. 14(c), the mean configure delay is shown as being lower above 2 hops compared to the uSDN-CSMA and uSDN-CONTIKIMAC protocols.

With reference to Fig. 15, the proportion of nodes joined over time is shown. The time for joining all nodes according to the present, Atomic-SDN, controller is shown as almost instantaneous, whereas the uSDN-CSMA and uSDN-CONTIKIMAC protocols exhibit slower times.

With reference to Fig. 16, the end-to-end PDR (%) and the radio duty cycle (%) are shown against the number of hops.

With reference to Fig. 17, the scalability of the present, Atomic-SDN, controller, is shown for each opportunity type. For instance, the time to scale the opportunity per hop is shown to be substantially linear.

With reference to Fig. 18, the controller 10 includes a processor 11 and a memory 13, storing instructions in the form of electronic data. The instructions are arranged to be executed by the processor 11, for performing the embodiment described above. The controller 10 is shown in communication with a node 15 of the network 17. The node 15 in communication with the controller 10 may be known as the control node.

While certain arrangements have been described, the arrangements have been presented by way of example only, and are not intended to limit the scope of protection. The inventive concepts described herein may be implemented in a variety of other forms. In addition, various omissions, substitutions and changes to the specific implementations described herein may be made without departing from the scope of protection defined in the following claims.

CLAIMS:

1. A controller for a low-power, wireless, software defined networking, SDN, architecture, comprising a processor and a memory storing instructions for execution by the processor, the instructions, when executed by the processor causing the controller to implement:

   a control layer arranged to detect a communication type for processing an application over the network, and translate the communication type into a plurality of phases chosen from a pre-defined set of phases to configure the nodes of the network to respond to the application according to the communication type.

2. The controller of Claim 1, wherein the communication type is selected from a list including: one-to-one communication; one-to-many communication; and many-to-one communication.

3. The controller of Claim 1, wherein the controller configures the nodes to respond to the application by selecting one mode from a list including: source mode; destination mode; and forwarder mode.

4. The controller of Claim 1, wherein the phases are scheduled in a time synchronous manner.

5. The controller of Claim 1, wherein each phase includes a generic flood primitive to configure each node of the network.

6. The controller of Claim 1, wherein each phase includes pre and post processing logic, the pre-processing logic arranged to modify the generic flood primitive based on the current performance of the network, and the post-processing logic arranged to detect a performance of the network in responding to the generic flood primitive.

7. The controller of Claim 1, wherein each phase includes a guard time period.

8. The controller of Claim 1, wherein each phase includes an offset time period.

9. The controller of Claim 1, further configured to divide a data processing epoch into a control plane and a subsequent data plane, wherein the controller is configured to transmit the phases to the network in the control plane, and configured to transmit the application data to the network in the data plane.

10. A low-power, wireless, software defined network, SDN, comprising:

a plurality of nodes; and

the controller of Claim 1 arranged to configure the nodes of the network to process the application over the network.

11. A method of processing data over a low-power, wireless, software defined networking, SDN, architecture, comprising:

detecting a communication type for processing an application over the network, and translating the communication type into a plurality of phases chosen from a pre-defined set of phases to configure the nodes of the network to respond to the application according to the communication type.

12. The method of Claim 11, wherein the communication type is selected from a list including: one-to-one communication; one-to-many communication; and many-to-one communication.

13. The method of Claim 11, wherein the nodes responding to the application includes acting in one mode selected from a list including: source mode; destination mode; and forwarder mode.

14. The method of Claim 11, further comprising scheduling the phases in a time synchronous manner.

15. The method of Claim 11, wherein each phase includes a generic flood primitive to configure each node of the network.

16. The method of Claim 11, further comprising detecting a performance of the network in responding to the generic floor primitive and modifying a subsequent flooding primitive based on the performance of the network.

17. The method of Claim 11, further comprising scheduling the phases to include a guard time period.

18. The method of Claim 11, further comprising scheduling the phases to include an offset time period.

19. The method of Claim 11, further comprising dividing a data processing epoch into a control plane and a subsequent data plane, transmitting the phases to the network in the control plane, and transmitting the application data to the network in the data plane.

20. Non-transitory memory storing computer program instructions for execution by a processor, the instructions, when executed by the processor, causing the processor to perform the method claimed in Claim 11.

ABSTRACT:

The present subject-matter provide a controller for a low-power, wireless, software defined networking, SDN, architecture. The controller comprises a processor and a memory storing instructions for execution by the processor. The instructions, when executed by the processor cause the controller to implement: a control layer arranged to detect a communication type for processing an application over the network, and translate the communication type into a plurality of phases chosen from a pre-defined set of phases to configure the nodes of the network to respond to the application according to the communication type.

Fig. 1

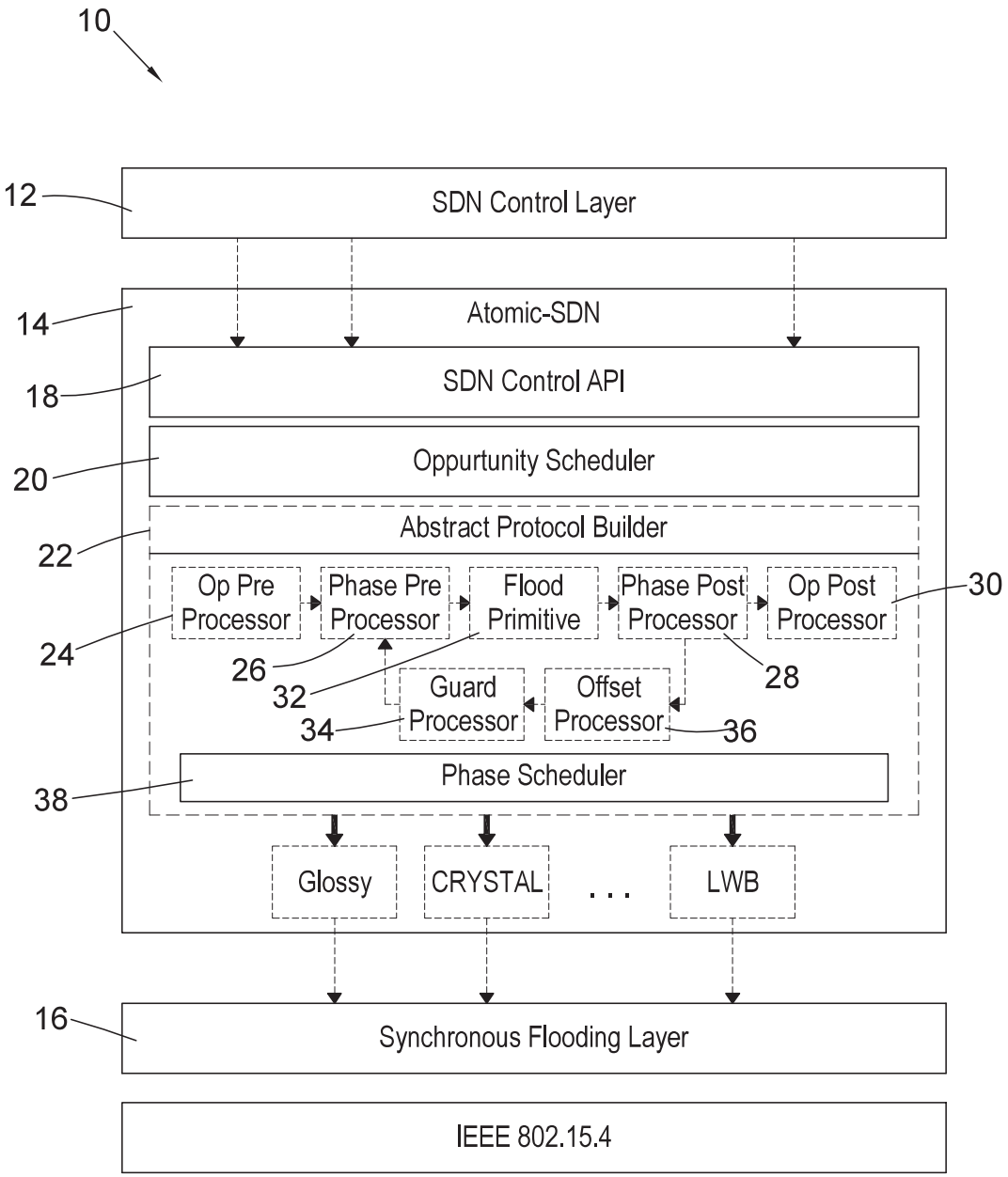

10
12
SDN Control Layer
14
Atomic-SDN
18
SDN Control API
20
Oppurtunity Scheduler
22
Abstract Protocol Builder
Op Pre Processor
Phase Pre Processor
Flood Primitive
Phase Post Processor
Op Post Processor
30
24
26
32
28
Guard Processor
Offset Processor
34
36
38
Phase Scheduler
Glossy
CRYSTAL
. . .
LWB
16
Synchronous Flooding Layer
IEEE 802.15.4

Fig. 1

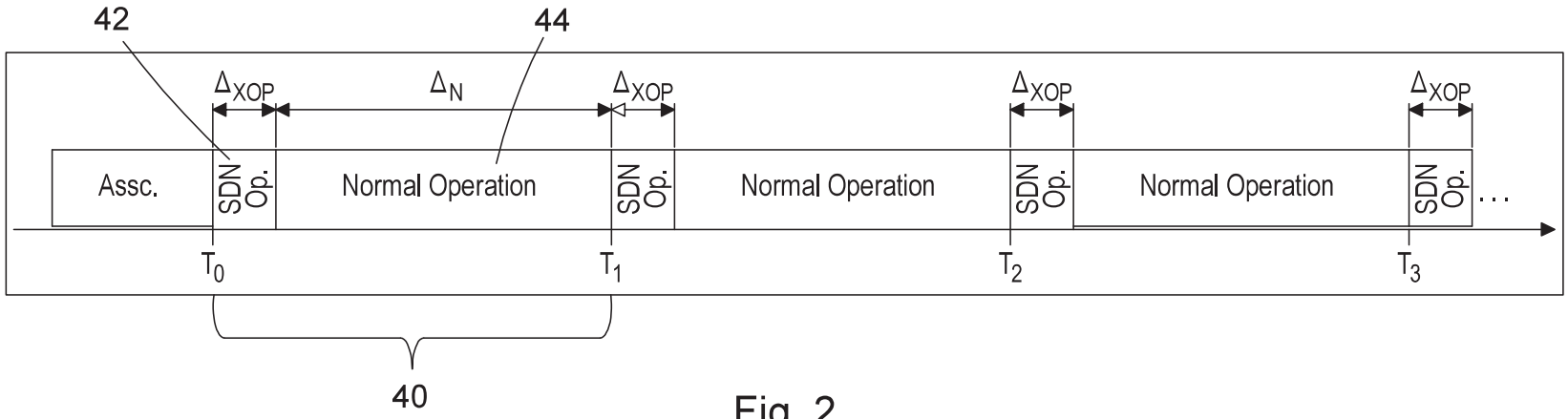

42
44
Δ_{XOP}
Δ_N
Δ_{XOP}
Δ_{XOP}
Δ_{XOP}
Assc.
SDN Op.
Normal Operation
SDN Op.
Normal Operation
SDN Op.
Normal Operation
SDN Op.
...
T_0
T_1
T_2
T_3
40

Fig. 2

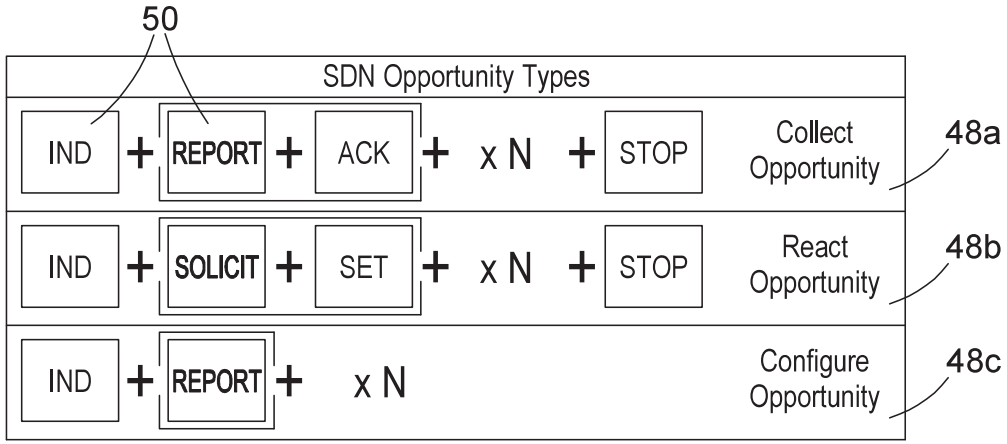

50
SDN Opportunity Types
IND + REPORT + ACK + x N + STOP
Collect Opportunity
48a
IND + SOLICIT + SET + x N + STOP
React Opportunity
48b
IND + REPORT + x N
Configure Opportunity
48c

Fig. 3

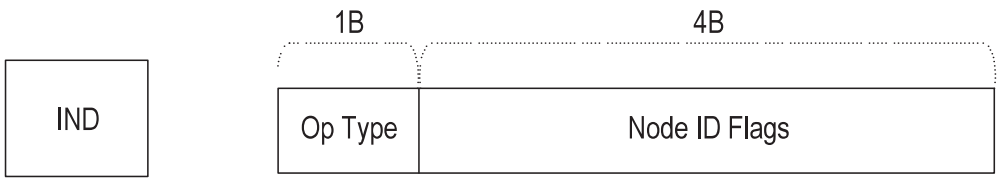


Fig. 4

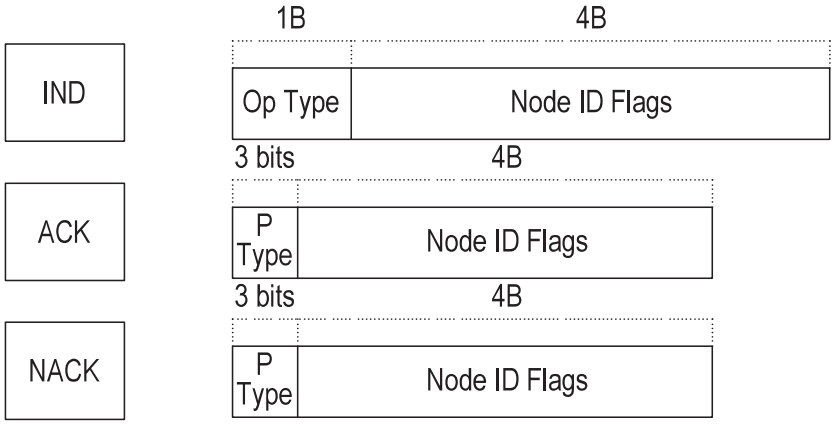


Fig. 5

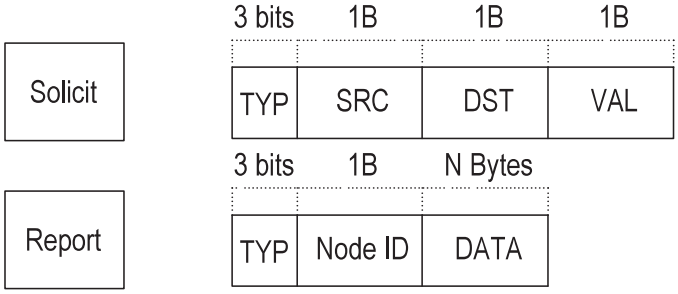


Fig. 6

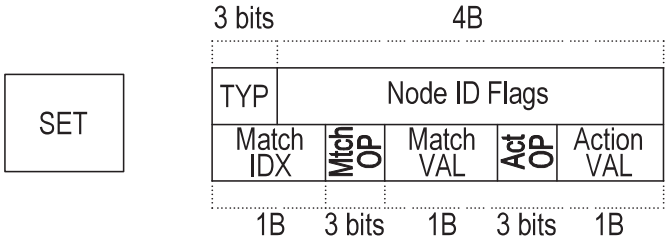


Fig. 7

Fig. 8

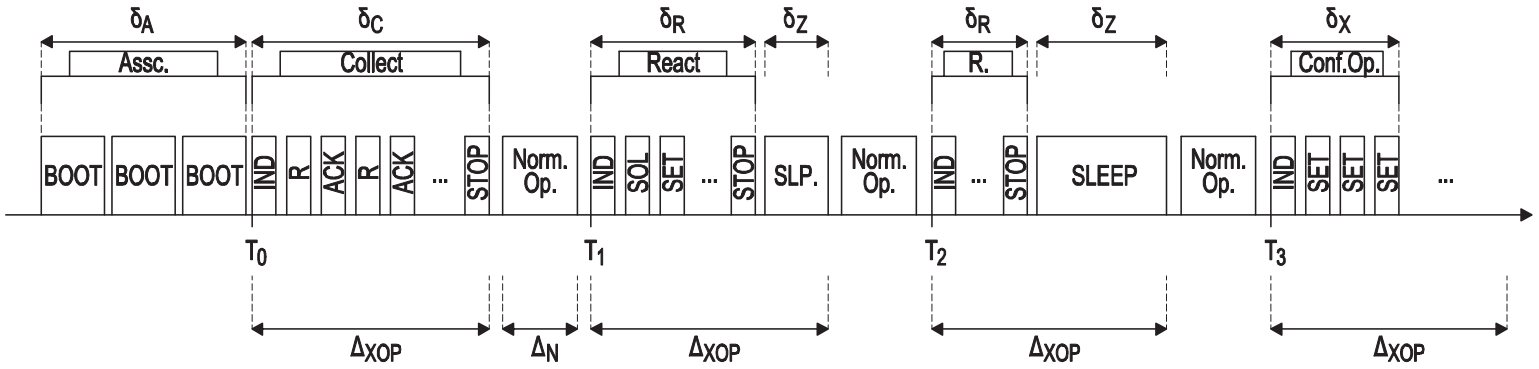


Fig. 9

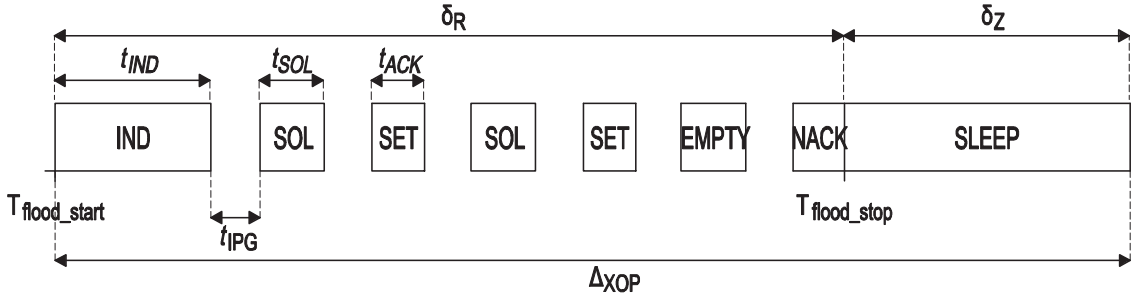


Fig. 10

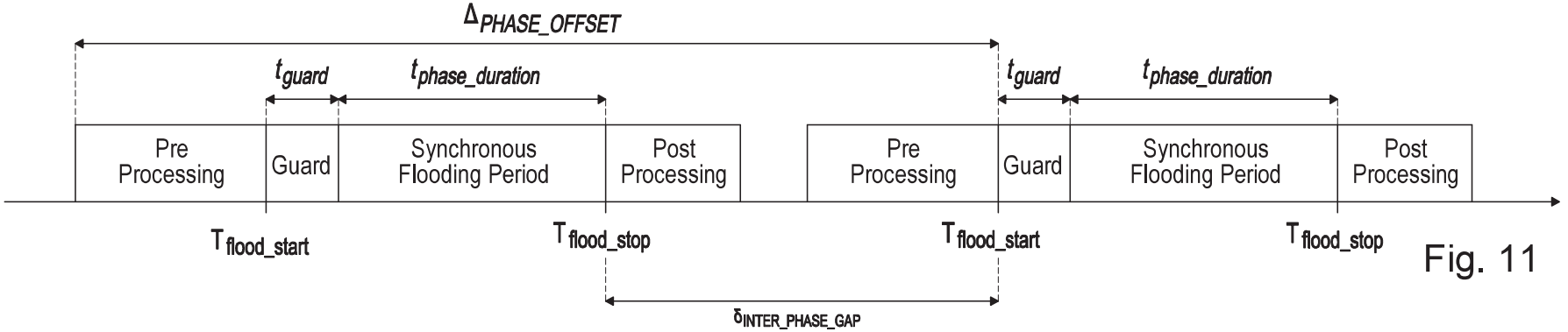


Fig. 11

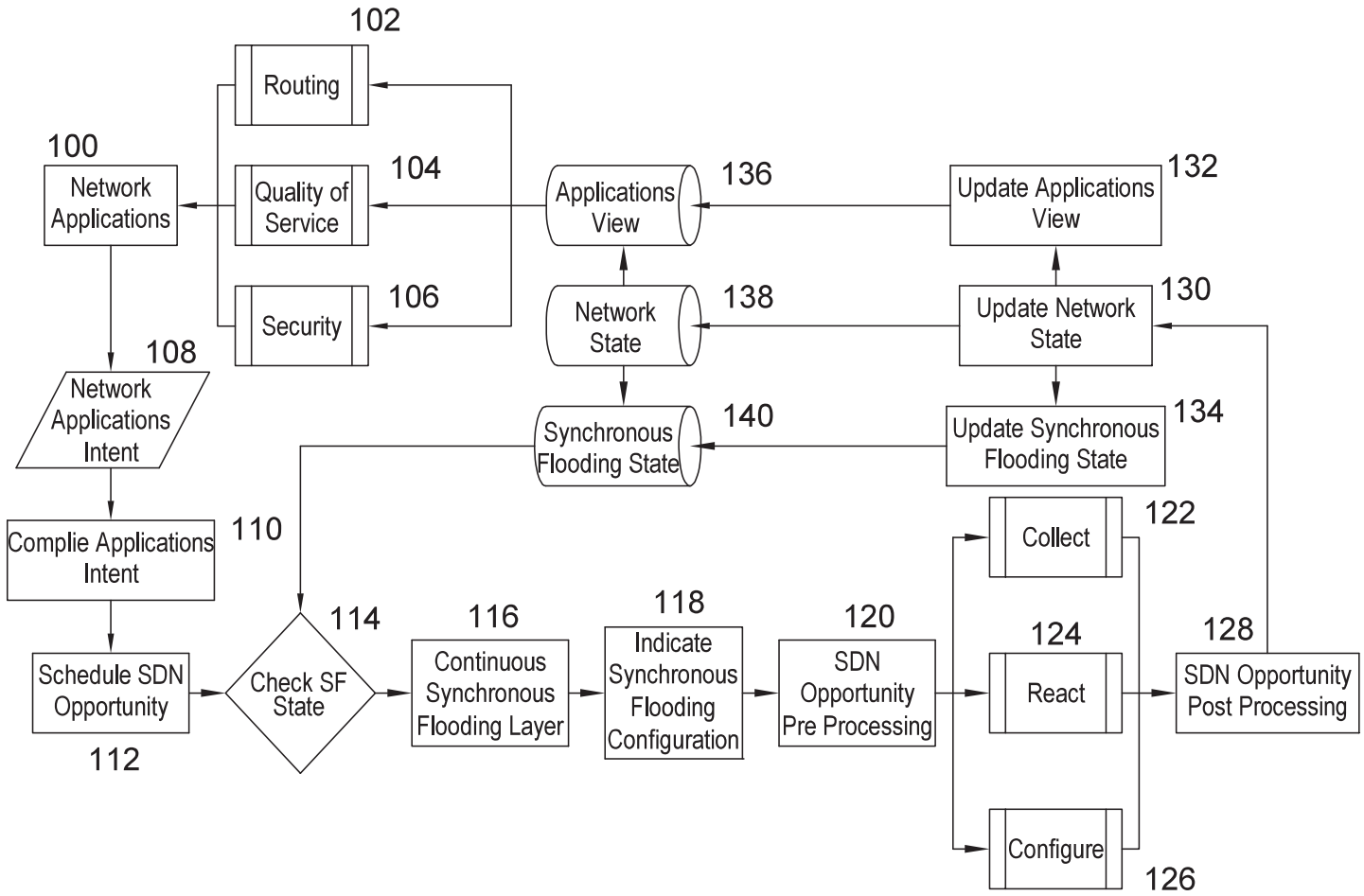

102
Routing
100
Network Applications
Quality of Service
104
Security
106
Applications View
136
Update Applications View
132
Network State
138
Update Network State
130
108
Network Applications Intent
Synchronous Flooding State
140
Update Synchronous Flooding State
134
Complie Applications Intent
110
Collect
122
114
116
118
120
124
128
Schedule SDN Opportunity
Check SF State
Continuous Synchronous Flooding Layer
Indicate Synchronous Flooding Configuration
SDN Opportunity Pre Processing
React
SDN Opportunity Post Processing
112
Configure
126

Fig. 12

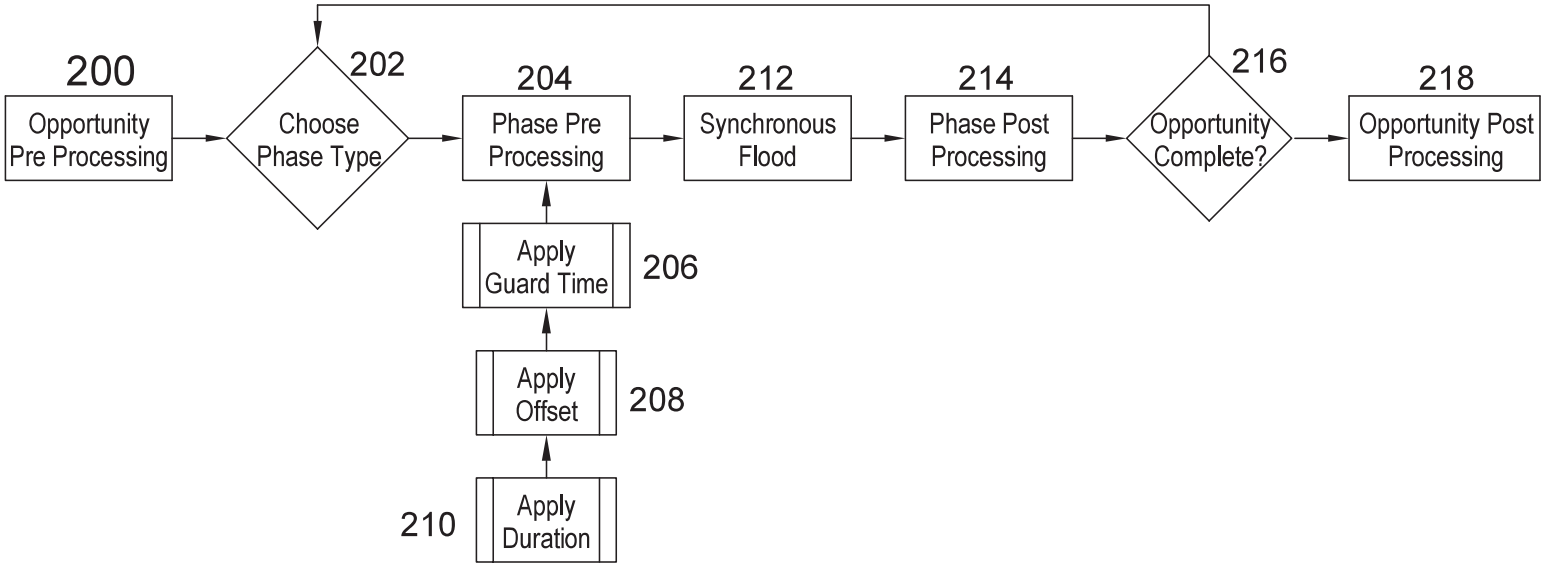

200
Opportunity Pre Processing
202
Choose Phase Type
204
Phase Pre Processing
212
Synchronous Flood
214
Phase Post Processing
216
Opportunity Complete?
218
Opportunity Post Processing
Apply Guard Time
206
Apply Offset
208
210
Apply Duration

Fig. 13

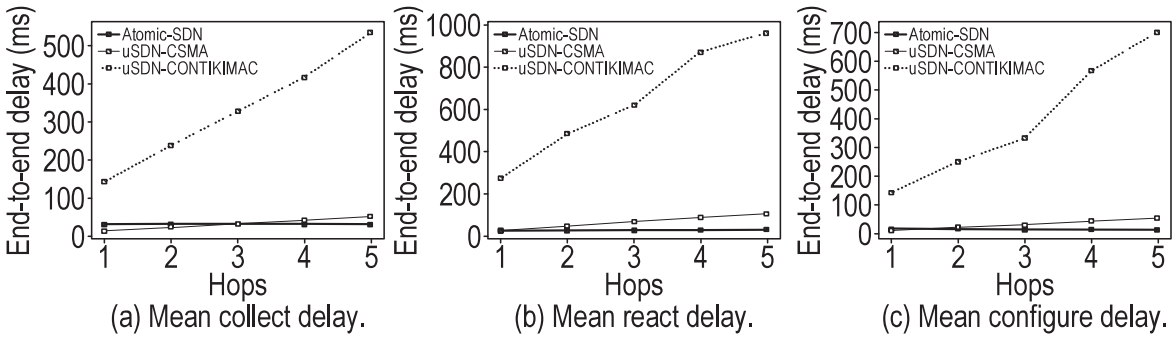

End-to-end delay (ms)
Atomic-SDN
uSDN-CSMA
uSDN-CONTIKIMAC
Hops
(a) Mean collect delay.
(b) Mean react delay.
(c) Mean configure delay.

Fig. 14

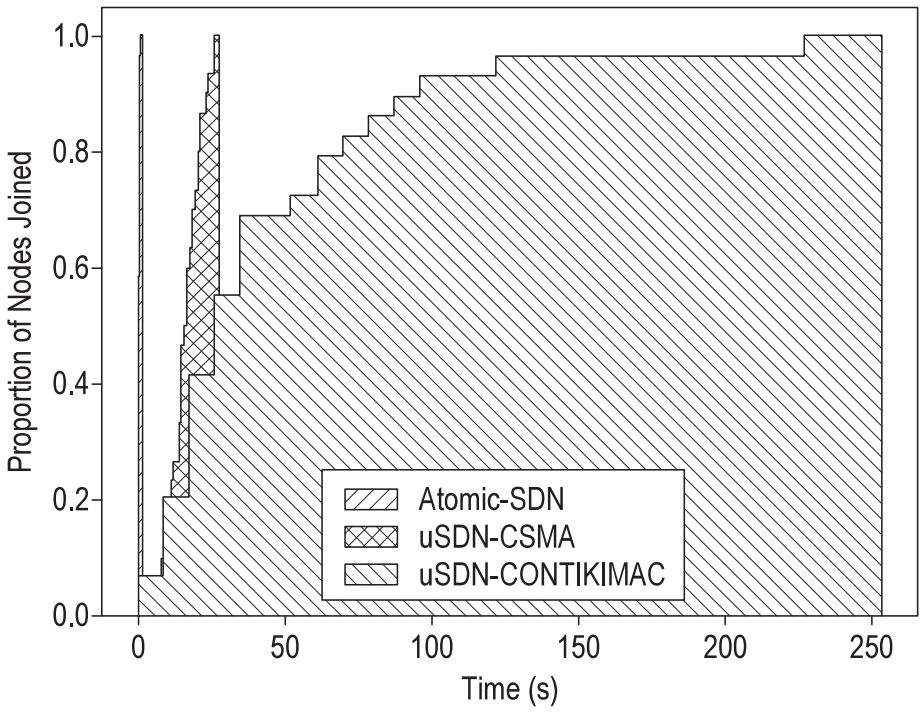

Proportion of Nodes Joined
Time (s)
Atomic-SDN
uSDN-CSMA
uSDN-CONTIKIMAC

Fig. 15

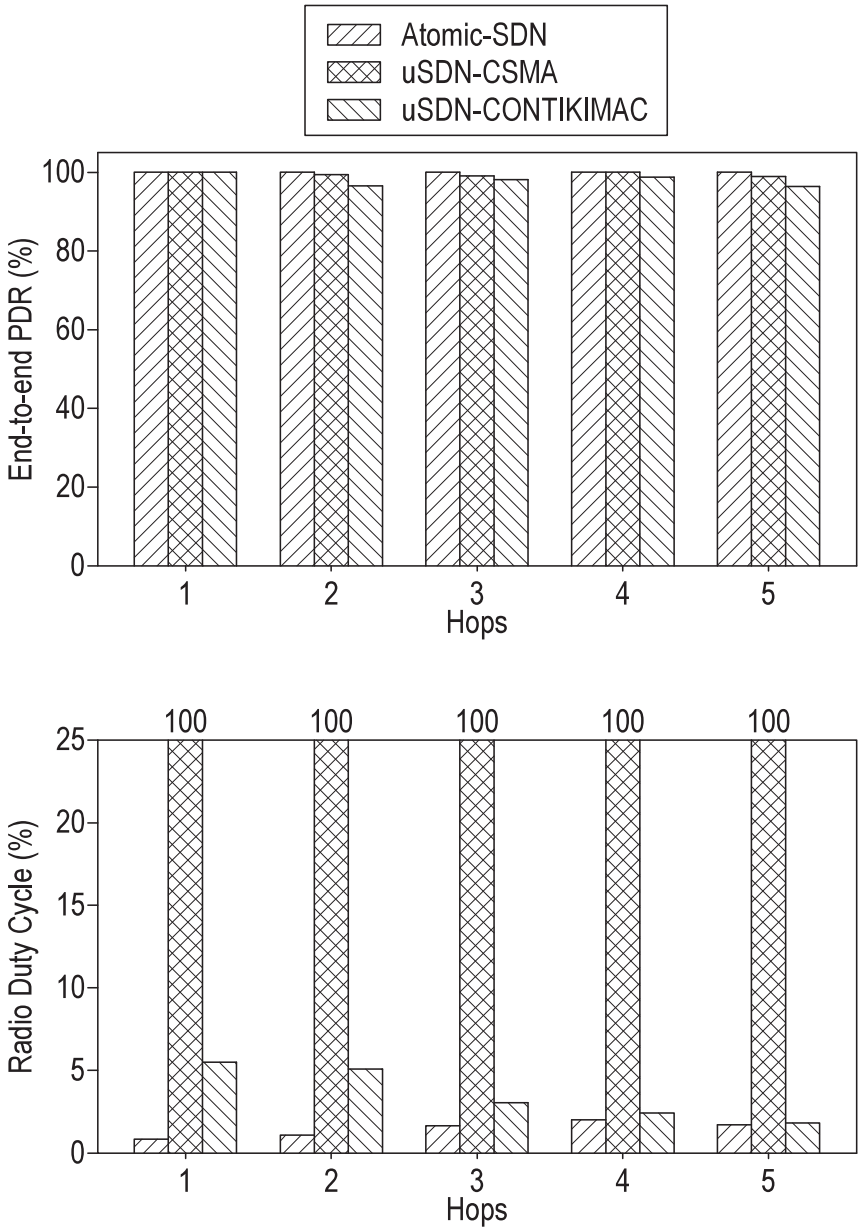

Atomic-SDN
uSDN-CSMA
uSDN-CONTIKIMAC
End-to-end PDR (%)
100
80
60
40
20
0
1
2
3
4
5
Hops
Radio Duty Cycle (%)
25
20
15
10
5
0
100
100
100
100
100
1
2
3
4
5
Hops

Fig. 16

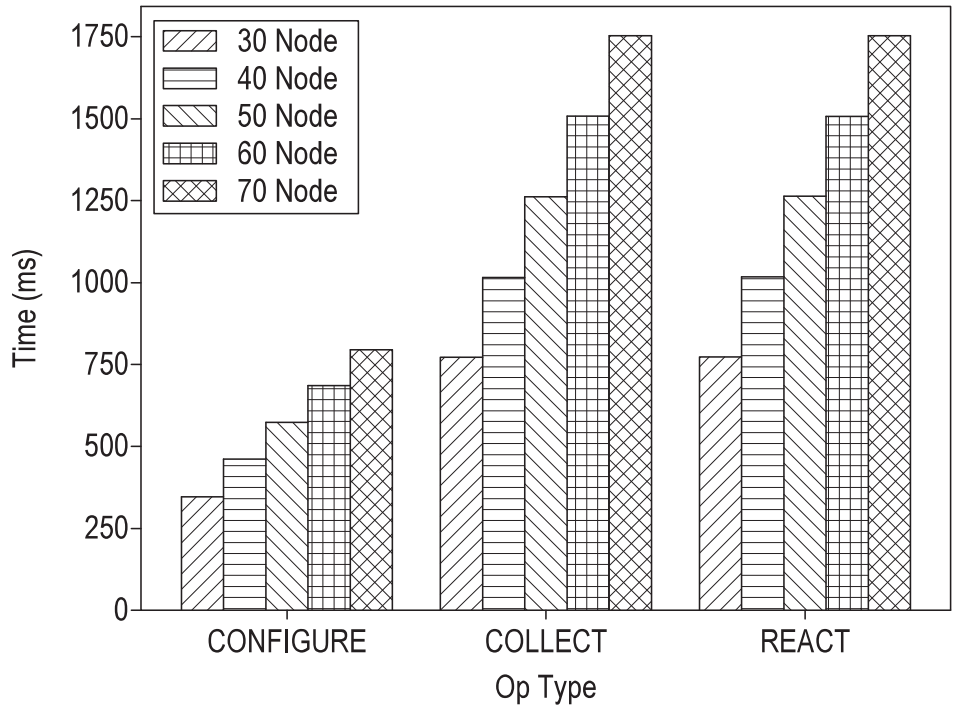

30 Node
40 Node
50 Node
60 Node
70 Node
1750
1500
1250
1000
750
500
250
0
Time (ms)
CONFIGURE
COLLECT
REACT
Op Type

Fig. 17

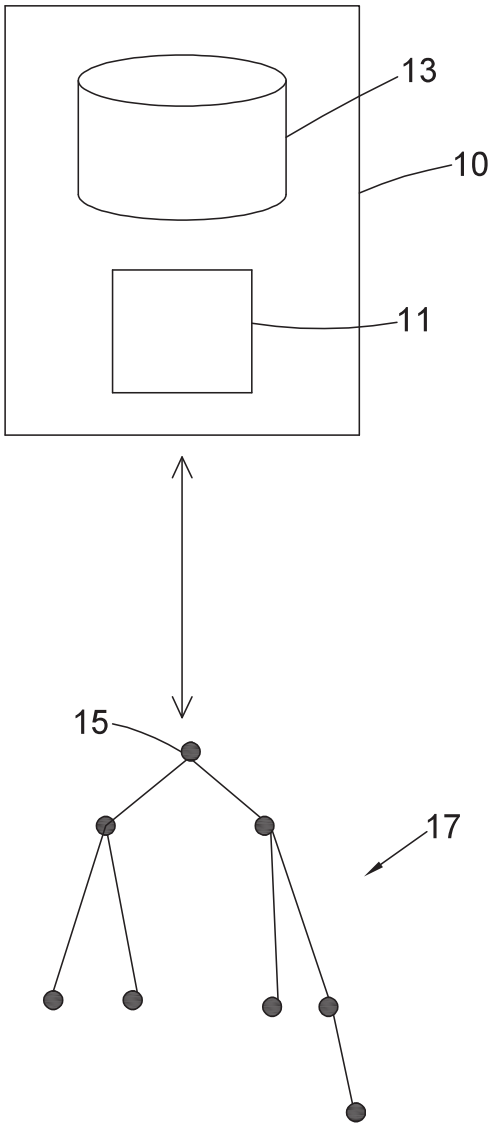


Fig. 18

BIBLIOGRAPHY

[1] IEEE, "IEEE Standard for Low-Rate Wireless Networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011, IEEE Std 802.15.4-2006, IEEE Std 802.15.4-2003)*, pp. 1–709, April 2016.

[2] P. Thubert, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4," Internet-Draft draft-ietf-6tisch-architecture-24, Internet Engineering Task Force, July 2019. Work in Progress.

[3] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, pp. 83–95, ACM, 2016.

[4] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, pp. 14–76, Jan 2015.

[5] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology." RFC 7426, Jan. 2015.

[6] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient Network Flooding and Time Synchronization with Glossy," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 73–84, April 2011.

[7] R. Lim, R. Da Forno, F. Sutton, and L. Thiele, "Competition: Robust Flooding Using Back-to-Back Synchronous Transmissions with Channel-Hopping," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2017.

[8] A. Escobar, F. Moreno, A. J. Cabrera, J. Garcia-Jimenez, F. J. Cruz, U. Ruiz, J. Klaue, A. Corona, D. Tati, and T. Meyerhoff, "Competition: BigBangBus," in *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, EWSN '18, pp. 213–214, 2018.

[9] U. Raza, Y. Jin, and M. Sooriyabandara, "Competition: Synchronous Transmissions Based Flooding for Dependable Internet of Things," in *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, pp. 278–279, 2017.

[10] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022 White Paper," 2018.
[Online; accessed 9-June-2019].

[11] R. Drath and A. Horch, "Industrie 4.0: Hit or Hype?," *IEEE Industrial Electronics Magazine*, vol. 8, pp. 56–58, June 2014.

[12] IEEE, "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN)," *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)*, pp. 1–700, June 2005.

[13] Lora Alliance, "LoRaWAN 1.1 Specification." https://lora-alliance.org/resource-hub/lorawanr-specification-v11, 2017.
[Online; accessed 7-June-2019].

[14] Landys+Gyr, "TEPCO and Landis+Gyr Sign Agreement to Explore Future Options for Leveraging IoT Network." https://www.landisgyr.eu/news/tepco-landisgyr-sign-agreement-explore-future-options-leveraging-iot-network-2/, 2017.
[Online; accessed 6-June-2019].

[15] Cisco, "Software-Defined Networking: Why We Like It and How We Are Building On It." https://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/cis13090_sdn_sled_white_paper.pdf, 2013.
[Online; accessed 6-June-2019].

[16] Albert Greenberg, "SDN for the Cloud." https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/keynote.pdf, 2015.
[Online; accessed 6-June-2019].

[17] Google, "Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization." https://www.usenix.org/system/files/conference/nsdi18/nsdi18-dalton.pdf, 2018.
[Online; accessed 6-June-2019].

[18] The Register, "BT berries Juniper's SDN kit in network architecture refresh." https://www.theregister.co.uk/2019/06/03/bt_juniper_sdn_network_architecture_refresh/, 2019.
[Online; accessed 6-June-2019].

[19] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing Software-Defined Networks: A Survey," *IEEE Access*, vol. 5, pp. 25487–25526, 2017.

[20] K. Sood, S. Yu, and Y. Xiang, "Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review," *IEEE Internet of Things Journal*, vol. 3, pp. 453–463, Aug 2016.

[21] I. T. Haque and N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 2713–2737, Fourthquarter 2016.

[22] S. Bera, S. Misra, and A. V. Vasilakos, "Software-Defined Networking for Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–1, 2017.

[23] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements," *IEEE access*, vol. 5, pp. 1872–1899, 2017.

[24] A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, T. H. Clausen, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks." RFC 6550, Mar. 2012.

[25] P. Dely, A. Kassler, and N. Bayer, "OpenFlow for Wireless Mesh Networks," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–6, July 2011.

[26] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 513–521, April 2015.

[27] R. Alves, D. Oliveira, G. Nez, and C. B. Margi, "IT-SDN: Improved Architecture for SDWSN," in *Proceedings of the XXXV Brazilian Symposium on Computer Networks and Distributed Systems, Belem, Brazil*, pp. 15–19, 2017.

[28] M. Brachmann, O. Landsiedel, and S. Santini, "Keep the Beat: On-The-Fly Clock Offset Compensation for Synchronous Transmissions in Low-Power Networks," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pp. 303–311, Oct 2017.

[29] "IEEE EWSN 2019 Results." `https://iti-testbed.tugraz.at/blog/page/21/ewsn-19-dependability-competition-final-results/#`, 2019.
[Online; accessed 31-July-2019].

[30] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–98, Apr. 2014.

[31] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[32] S. Shenker, "The Future of Networking, and the Past of Protocols?." `https://www.youtube.com/watch?v=YHeyuD89n1Y`, 2016.
[Online; accessed 10-May-2019].

[33] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[34] "OpenFlow Switch Specification." `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf`, 2015.
[Online; accessed 10-May-2019].

[35] H. Song, "Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, (New York, NY, USA), pp. 127–132, ACM, 2013.

[36] B. Belter, D. Parniewicz, L. Ogrodowczyk, A. Binczewski, M. Stroiński, V. Fuentes, J. Matias, M. Huarte, and E. Jacob, "Hardware Abstraction Layer as an SDN-enabler for Non-OpenFlow Network Equipment," in *EWSDN*, pp. 117–118, 2014.

[37] B. Pfaff and B. Davie, "The Open vSwitch Database Management Protocol." RFC 7047, Dec. 2013.

[38] M. Smith, R. E. Adams, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, "OpFlex Control Protocol," Internet-Draft draft-smith-opflex-03, Internet Engineering Task Force, Apr. 2016.
Work in Progress.

[39] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 44–51, Apr. 2014.

[40] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "A Survey of Programmable Networks," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, pp. 7–23, 1999.

[41] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A Survey of Active Network Research," *IEEE communications Magazine*, vol. 35, no. 1, pp. 80–86, 1997.

[42] J. M. Halpern, R. HAAS, A. Doria, L. Dong, W. Wang, H. M. Khosravi, J. H. Salim, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Protocol Specification." RFC 5810, Mar. 2010.

[43] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending Networking into the Virtualization Layer," in *Hotnets*, 2009.

[44] Y. Rekhter, S. Hares, and T. Li, "A Border Gateway Protocol 4 (BGP-4)." RFC 4271, Jan. 2006.

[45] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case, "Simple Network Management Protocol (SNMP)." RFC 1157, May 1990.

[46] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, "Network Configuration Protocol (NETCONF)." RFC 6241, June 2011.

[47] J. Vasseur and J.-L. L. Roux, "Path Computation Element (PCE) Communication Protocol (PCEP)." RFC 5440, Mar. 2009.

[48] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core." RFC 6120, Mar. 2011.

[49] A. Viswanathan, E. C. Rosen, and R. Callon, "Multiprotocol Label Switching Architecture." RFC 3031, Jan. 2001.

[50] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." RFC 7348, Aug. 2014.

[51] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[52] P4, "P4 Language Consortium." `https://p4.org/`, 2019.
[Online; accessed 10-May-2019].

[53] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 655–685, Firstquarter 2016.

[54] S. Vinoski, "Advanced Message Queuing Protocol," *IEEE Internet Computing*, no. 6, pp. 87–89, 2006.

[55] IEEE, "IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7," *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)*, pp. 1–3951, Jan 2018.

[56] W. Zhou, L. Li, M. Luo, and W. Chou, "REST API design patterns for SDN northbound API," in *2014 28th international conference on advanced information networking and applications workshops*, pp. 358–365, IEEE, 2014.

[57] The Open Networking Foundation, "Open Networking Foundation North Bound Interface Working Group (NBI-WG) Charter." `https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf`, 2015.
[Online; accessed 12-May-2019].

[58] Linux Foundation, "OpenDaylight." `https://www.opendaylight.org/`.
[Online; accessed 10-May-2019].

[59] ONOS Project, "ONOS." `https://onosproject.org/`.
[Online; accessed 10-May-2019].

[60] J. van de Belt, H. Ahmadi, and L. Doyle, "Defining and Surveying Wireless Link and Network Virtualization," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2017.

[61] C. J. Bernardos, A. Rahman, J.-C. Zúñiga, L. M. Contreras, P. A. Aranda, and P. Lynch, "Network Virtualization Research Challenges." RFC 8568, Apr. 2019.

[62] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, *et al.*, "Carving Research Slices out of your Production Networks with OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 129–130, 2010.

[63] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-Of-The-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[64] A. Escobar-Molero, "Improving Reliability and Latency of Wireless Sensor Networks Using Concurrent Transmissions," *at-Automatisierungstechnik*, vol. 67, no. 1, pp. 42–50, 2019.

[65] K. Leentvaar and J. Flint, "The Capture Effect in FM Receivers," *IEEE Transactions on Communications*, vol. 24, no. 5, pp. 531–539, 1976.

[66] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)." RFC 7252, June 2014.

[67] P. Kamalinejad, C. Mahapatra, Z. Sheng, S. Mirabbasi, V. C. M. Leung, and Y. L. Guan, "Wireless Energy Harvesting for the Internet of Things," *IEEE Communications Magazine*, vol. 53, pp. 102–108, June 2015.

[68] F. K. Shaikh and S. Zeadally, "Energy Harvesting in Wireless Sensor Networks: A Comprehensive Review," *Renewable and Sustainable Energy Reviews*, vol. 55, pp. 1041–1054, 2016.

[69] C. I. D. E. 2017/1483, "Commission Implementing Decision (EU) 2017/1483 of 8 August 2017 amending Decision 2006/771/EC on harmonisation of the radio spectrum for use by short-range devices and repealing Decision 2006/804/EC." `https://eur-lex.europa.eu/eli/dec_impl/2017/1483/oj`, 2017.
[Online; accessed 21-May-2019].

[70] IEEE, "IEEE Standard for Telecommunications and Information Exchange Between Systems– LAN/MAN Specific Requirements– Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPAN)," *IEEE Std 802.15.4-2003*, pp. 1–680, Oct 2003.

[71] "ZigBee Specification Document." `https://www.zigbee.org/download/standards-zigbee-specification/`, 2012.
[Online; accessed 13-May-2019].

[72] "System Engineering Guidelines IEC 62591 WirelessHART." `https://www.emerson.com/documents/automation/engineering-guide-system-engineering-guidelines-iec-62591-wirelesshart-en-79900.pdf`, 2016.
[Online; accessed 13-May-2019].

[73] "ANSI/ISA-100.11a-2011 Wireless Systems for Industrial Automation: Process Control and Related Applications." `https://www.isa.org/store/products/product-detail/?productId=118261`, 2011.
[Online; accessed 14-May-2019].

[74] "Wi-SUN Alliance." `https://www.wi-sun.org/`, 2019.
[Online; accessed 22-May-2019].

[75] "TI RF Range Estimator." `http://www.ti.com/tool/RF-RANGE-ESTIMATOR#`, 2015.
[Online; accessed 7-June-2019].

[76] IEEE, "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pp. 1–320, Sep. 2006.

[77] C.-H. Liao, Y. Katsumata, M. Suzuki, and H. Morikawa, "Revisiting the So-Called Constructive Interference in Concurrent Transmission," in *Local Computer Networks (LCN), 2016 IEEE 41st Conference on*, pp. 280–288, IEEE, 2016.

[78] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Concurent Transmisions for Multi-hop Bluetooth 5," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, EWSN '19, 2019.

[79] B. A. Nahas, A. Escobar-Molero, J. Klaue, S. Duquennoy, and O. Landsiedel, "BlueFlood: Concurrent Transmissions for Multi-Hop Bluetooth 5–Modeling and Evaluation," *arXiv preprint arXiv:2002.12906*, 2020.

[80] M. Zimmerling, L. Mottola, and S. Santini, "Synchronous transmissions in low-power wireless: A survey of communication protocols and network services," *arXiv preprint arXiv:2001.08557*, 2020.

[81] N. Salman, I. Rasool, and A. H. Kemp, "Overview of the IEEE 802.15.4 Standards Family for Low Rate Wireless Personal Area Networks," in *2010 7th International Symposium on Wireless Communication Systems*, pp. 701–705, Sep. 2010.

[82] T. Junjalearnvong, R. Okumura, K. Mizutani, and H. Harada, "Performance Evaluation of Multi-hop Network Configuration for Wi-SUN FAN Systems," in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–6, Jan 2019.

[83] IEEE, "IEEE Standard for Local and metropolitan area networks– Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer," *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pp. 1–225, April 2012.

[84] G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks." RFC 4944, Sept. 2007.

[85] P. Thubert and J. Hui, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks." RFC 6282, Sept. 2011.

[86] C. Bormann, Z. Shelby, S. Chakrabarti, and E. Nordmark, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)." RFC 6775, Nov. 2012.

[87] E. Kim, D. Kaspar, C. Gomez, and C. Bormann, "Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing." RFC 6606, May 2012.

[88] T. Watteyne, C. Bormann, and P. Thubert, "LLN Minimal Fragment Forwarding," Internet-Draft draft-ietf-6lo-minimal-fragment-01, Internet Engineering Task Force, Mar. 2019.

Work in Progress.

[89] O. Gnawali and P. Levis, "The Minimum Rank with Hysteresis Objective Function." RFC 6719, Sept. 2012.

[90] P. Thubert, "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)." RFC 6552, Mar. 2012.

[91] M. Nixon and T. Round Rock, "A Comparison of WirelessHART and ISA100.11a," *Whitepaper, Emerson Process Management*, pp. 1–36, 2012.

[92] N. Finn, P. Thubert, B. Varga, and J. Farkas, "Deterministic Networking Architecture," Internet-Draft draft-ietf-detnet-architecture-13, Internet Engineering Task Force, May 2019.
Work in Progress.

[93] L. G. Roberts, "ALOHA Packet System with and Without Slots and Capture," *SIGCOMM Comput. Commun. Rev.*, vol. 5, pp. 28–42, Apr. 1975.

[94] T. Chang, T. Watteyne, X. Vilajosana, and P. H. Gomes, "Constructive Interference in 802.15.4: A Tutorial," *IEEE Communications Surveys & Tutorials*, 2018.

[95] U. Raza, A. Stanoev, C. Khoury, A.-I. Pop, and M. Sooriyabandara, "Synchronous Transmissions Based Flooding over Bluetooth 5.0 for Industrial Wireless Applications," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, EWSN '19, 2019.

[96] D. Vecchia, P. Corbalán, T. Istomin, and G. P. Picco, "Playing with Fire: Exploring Concurrent Transmissions in Ultra-wideband Radios," in *Proceedings of the 18th IEEE International Conference on Sensing, Communication and Networking*, SECON '19, 2019.

[97] T. Istomin, M. Trobinger, A. L. Murphy, and G. P. Picco, "Interference-rRsilient Ultra-low Power Aperiodic Data Collection," in *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '18, pp. 84–95, IEEE Press, 2018.

[98] C. Herrmann, F. Mager, and M. Zimmerling, "Mixer: Efficient Many-to-All Broadcast in Dynamic Wireless Mesh Networks," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pp. 145–158, ACM, 2018.

[99] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power Wireless Bus," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pp. 1–14, ACM, 2012.

[100] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, (New York, NY, USA), pp. 1:1–1:14, ACM, 2013.

[101] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Network-Wide Consensus Utilizing the Capture Effect in Low-power Wireless Networks," in *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys 2017)*, p. 14, 2017.

[102] R. Jacob, J. Baechli, R. Da Forno, and L. Thiele, "Synchronous Transmissions Made Easy: Design Your Network Stack with Baloo," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, EWSN '19, 2019.

[103] T. Luo, H. P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks," *IEEE Communications Letters*, vol. 16, pp. 1896–1899, November 2012.

[104] T. Luo, H. P. Tan, P. C. Quan, Y. W. Law, and J. Jin, "Enhancing Responsiveness and Scalability for OpenFlow networks via Control-Message Quenching," in *2012 International Conference on ICT Convergence (ICTC)*, pp. 348–353, Oct 2012.

[105] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software Defined Wireless Networks: Unbridling SDNs," in *2012 European Workshop on Software Defined Networking*, pp. 1–6, Oct 2012.

[106] H. Huang, P. Li, S. Guo, and W. Zhuang, "Software-Defined Wireless Mesh Networks: Architecture and Traffic Orchestration," *IEEE network*, vol. 29, no. 4, pp. 24–30, 2015.

[107] R. Huang, X. Chu, J. Zhang, and Y. H. Hu, "Energy-Efficient Monitoring in Software Defined Wireless Sensor Networks Using Reinforcement Learning: A Prototype," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, p. 360428, 2015.

[108] T. Theodorou and L. Mamatas, "CORAL-SDN: A software-defined networking solution for the Internet of Things," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–2, Nov 2017.

[109] T. Theodorou and L. Mamatas, "Software Defined Topology Control Strategies for the Internet of Things," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 236–241, IEEE, 2017.

[110] N. Q. Hieu, N. H. Thanh, T. T. Huong, N. Q. Thu, and H. Van Quang, "Integrating Trickle Timing in Software Defined WSNs for Energy Efficiency," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, pp. 75–80, IEEE, 2018.

[111] A. De Gante, M. Aslan, and A. Matrawy, "Smart Wireless Sensor Network Management Based on Software-Defined Networking," in *Communications (QBSC), 2014 27th Biennial Symposium on*, pp. 71–75, IEEE, 2014.

[112] B. T. de Oliveira, L. B. Gabriel, and C. B. Margi, "TinySDN: Enabling Multiple Controllers for Software-Defined Wireless Sensor Networks," *IEEE Latin America Transactions*, vol. 13, pp. 3690–3696, Nov 2015.

[113] B. T. de Oliveira and C. B. Margi, "Distributed control plane architecture for software-defined Wireless Sensor Networks," in *Consumer Electronics (ISCE), 2016 IEEE International Symposium on*, pp. 85–86, IEEE, 2016.

[114] I. Haque, M. Nurujjaman, J. Harms, and N. Abu-Ghazaleh, "SDSense: An Agile and Flexible SDN-Based Framework for Wireless Sensor Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, pp. 1866–1876, Feb 2019.

[115] L. L. Bello, A. Lombardo, S. Milardo, G. Patti, and M. Reno, "Software-Defined Networking for Dynamic Control of Mobile Industrial Wireless Sensor Networks," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 290–296, IEEE, 2018.

[116] E. Municio, J. Marquez-Barja, S. Latré, and S. Vissicchio, "Whisper: Programmable and Flexible Control on Industrial IoT Networks," *Sensors*, vol. 18, no. 11, p. 4048, 2018.

[117] A. Dunkels, "Rime - A Lightweight Layered Communication Stack for Sensor Networks," in *European Conference on Wireless Sensor Networks (EWSN)*, 2007.

[118] P. Levis, T. H. Clausen, O. Gnawali, J. Hui, and J. Ko, "The Trickle Algorithm." RFC 6206, Mar. 2011.

[119] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, *et al.*, "TinyOS: An Operating System for Sensor Networks," in *Ambient intelligence*, pp. 115–148, Springer, 2005.

[120] Q. Wang, X. Vilajosana, and T. Watteyne, "6TiSCH Operation Sublayer (6top) Protocol (6P)." RFC 8480, Nov. 2018.

[121] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," 2011.

[122] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, pp. 455–462, Nov 2004.

[123] IEEE, "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, Dec 2016.

[124] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-Defined Networks," *Hot-ICE*, vol. 12, pp. 1–6, 2012.

[125] P. Thubert, M. R. Palattella, and T. Engel, "6TiSCH centralized scheduling: When SDN meet IoT," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 42–47, Oct 2015.

[126] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration." RFC 8180, May 2017.

[127] P. Thubert, D. Cavalcanti, X. Vilajosana, and C. Schmitt, "Reliable and Available Wireless Technologies," Internet-Draft draft-thubert-raw-technologies-03, Internet Engineering Task Force, July 2019.
Work in Progress.

[128] T. Chang, M. Vučinić, X. Vilajosana, S. Duquennoy, and D. Dujovne, "6TiSCH Minimal Scheduling Function (MSF)," Internet-Draft draft-ietf-6tisch-msf-05, Internet Engineering Task Force, July 2019.
Work in Progress.

[129] M. Trobinger, T. Istomin, A. L. Murphy, and G. P. Picco, "Competition: CRYSTAL Clear: Making Interference Transparent," in *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, EWSN '18, pp. 217–218, 2018.

[130] A. Escobar, J. Garcia-Jimenez, F. J. Cruz, J. Klaue, A. Corona, and D. Tati, "Competition: RedFixHop with Channel Hopping," in *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, pp. 264–265, Junction Publishing, 2017.

[131] B. Al Nahas and O. Landsiedel, "Competition: Towards Low-Power Wireless Networking That Survives Interference with Minimal Latency," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2017.

[132] O. Bergmann, C. Bormann, S. Gerdes, and H. Chen, "Constrained-Cast: Source-Routed Multicast for RPL," Oct. 2017.
Work in Progress.

[133] L. Tang, Y. Sun, O. Gurewitz, and D. B. Johnson, "EM-MAC: A Dynamic Multichannel Energy-Efficient MAC Protocol for Wireless Sensor Networks," in *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, p. 23, ACM, 2011.

[134] B. A. Nahas, S. Duquennoy, V. Iyer, and T. Voigt, "Low-Power Listening Goes Multi-channel," in *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 2–9, May 2014.

[135] T. Watteyne, S. Lanzisera, A. Mehta, and K. S. J. Pister, "Mitigating Multipath Fading through Channel Hopping in Wireless Sensor Networks," in *2010 IEEE International Conference on Communications*, pp. 1–5, May 2010.

[136] M. Baddeley, A. Stanoev, U. Raza, Y. Jin, and M. Sooryiabandara, "Competition: Adaptive Software Defined Scheduling of Low-Power Wireless Networks," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, 2019.

[137] C. Noda, C. M. Pérez-Penichet, B. Seeber, M. Zennaro, M. Alves, and A. Moreira, "On the Scalability of Constructive Interference in Low-Power Wireless Networks," in *European Conference on Wireless Sensor Networks*, pp. 250–257, 2015.

[138] Y. Wang, Y. He, X. Mao, Y. Liu, and X. Li, "Exploiting Constructive Interference for Scalable Flooding in Wireless Networks," *IEEE/ACM Transactions on Networking*, vol. 21, pp. 1880–1889, Dec 2013.

[139] F. Mager, D. Baumann, R. Jacob, L. Thiele, S. Trimpe, and M. Zimmerling, "Demo Abstract: Fast Feedback Control and Coordination with Mode Changes for Wireless Cyber-Physical Systems," in *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 340–341, IEEE, 2019.

[140] A. Aijaz, A. Stanoev, and M. Sooriyabandara, "Toward Real-Time Wireless Control of Mobile Platforms for Future Industrial Systems," *arXiv preprint arXiv:1907.01979*, 2019.

[141] S. Duquennoy, A. Elsts, B. Al Nahas, and G. Oikonomo, "TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 11–18, IEEE, 2017.

[142] A. Elsts, S. Duquennoy, X. Fafoutis, G. Oikonomou, R. Piechocki, and I. Craddock, "Microsecond-Accuracy Time Synchronization Using the IEEE 802.15.4 TSCH Protocol," in *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, pp. 156–164, IEEE, 2016.